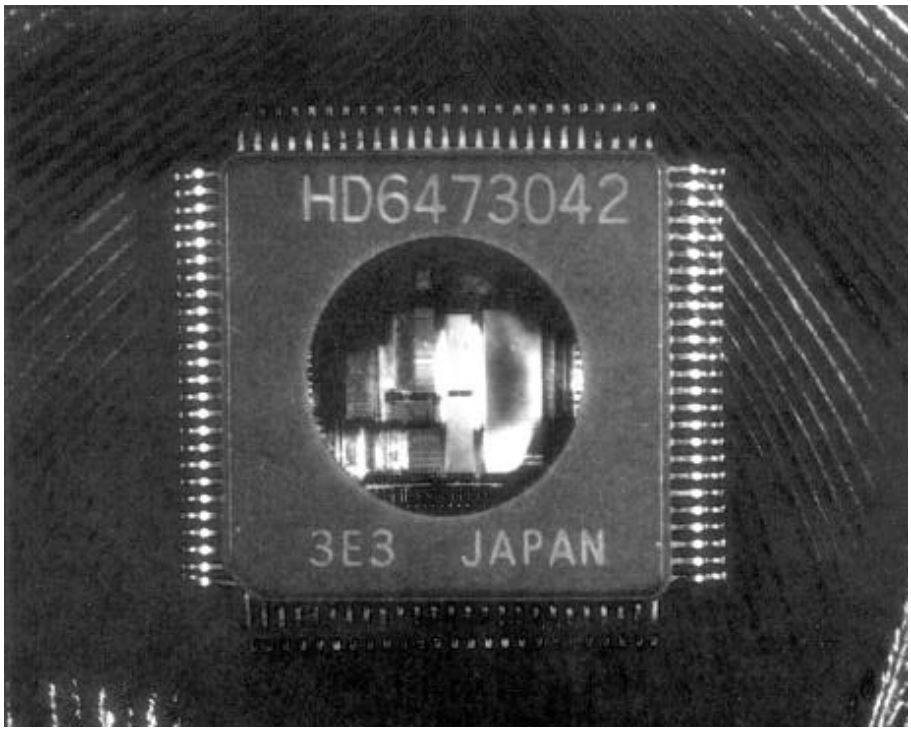
H8 / 300H

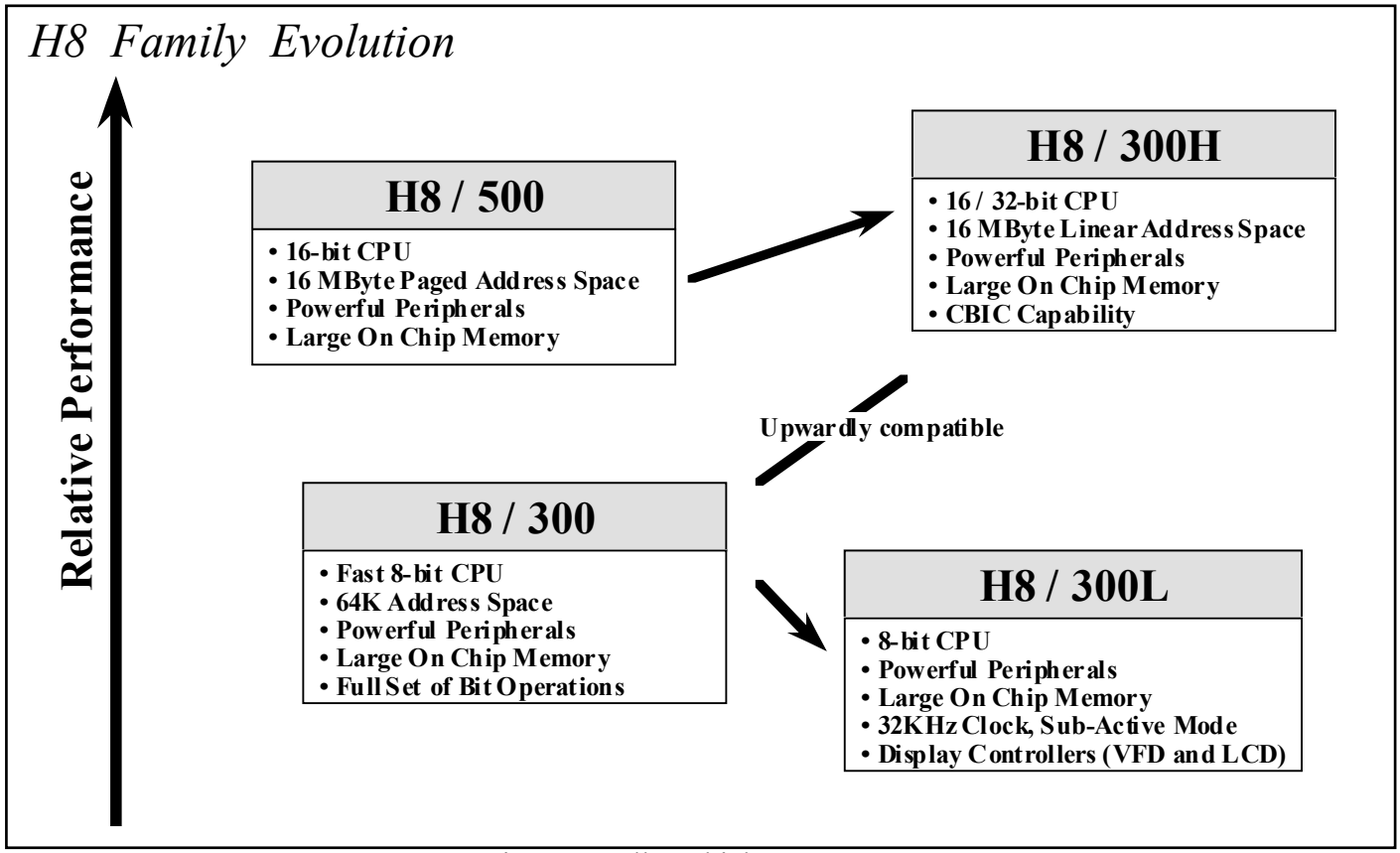
high performance 16 / 32-bit microcontrollers



Notes

1. This catalogue may wholly or partially be subject to change without notice.
2. This catalogue neither ensures the enforcement of any industrial properties on other rights, nor sanctions the enforcement right thereof.
   Examples of circuits given in this catalogue are only for a better understanding of the products. Therefore, Hitachi will not be responsible for any accidents or problems caused during operation.
3. All rights reserved: No-one is permitted to reproduce or duplicate, in any form, the whole or a part of this manual without Hitachi's permission.

INDEX



Introduction

As 16-bit microcontrollers become widely available, the demands placed on them by today's embedded systems is driving their development in the direction of much higher performance, lower power and more integration. This comes about from the continuous need to improve the equipment in which 16-bit technology is incorporated.

As mobile phones migrate to the new digital technologies their designers are demanding more processing power using smaller batteries and taking less PCB space.

The new consumer-minded PC printer technology needs extensive areas of low cost memory plus expanding processing power and specialised peripherals.

Low cost inverter controllers for AC motors require a microcontroller which can generate the motor drive waveforms, without resorting to any external circuits.

And in today's competitive markets for end equipment, the development cycle of the increasingly complex product must remain short. Thus engineers are now focusing more than ever on the means which can make the development team as productive and responsive as possible.

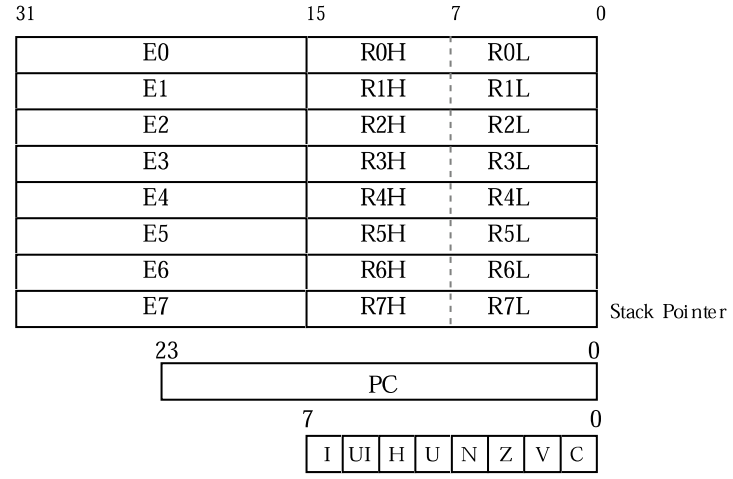
These include the use of a High Level Language (HLL), powerful debuggers and HLL friendly emulators. And as the complexity of these embedded systems increases, so does the need to use some form of real time operating system (RTOS) to ensure that the investment made in code development can be used for future projects.

H8 / 300H - The 16-bit Answer

With it's great experience in the development and marketing of 4, 8 and 16-bit microcontrollers for a wide range of applications, Hitachi listened to the demands of embedded systems developers when the H8/300H range of 16-bit microcontrollers was conceived. This powerful new range combines an upgrade path for users of the popular 8-bit H8/300 microcontroller with a totally new standard in 16-bit performance, to provide the answer to a whole generation of end equipments.

The H8/300H answers the need for more processing performance with a 1.9 Dhrystone MIPS, 16 / 32-bit CPU core. The need for large, complex programs and huge areas of data is answered with a 16MBytes linear address space. And where applications need on-chip memory, the H8 / 300H replies

Figure 1 - H8 / 300H CPU Core



with a market leading 128KBytes ROM (OTP or Mask) / 4KBytes RAM device.

Many integrated peripherals further reduce the chip count required to realise a system and, to keep power dissipation to the minimum, 3V operation can be specified for all variants. And finally to ensure that the size of the microcontroller does not limit it's use in size critical systems, the H8/300H is manufactured in space saving packages. These include a 100-pin thin QFP which is 14mm across it's gull wings and only 1.2mm in height.

Hitachi also recognises the importance of an effective design environment for embedded controllers. Therefore, a full development suite can be provided by Hitachi, or by a growing number of third party vendors. ANSI C Compilers, a real time operating system, C level debuggers running as simulators on the development host or in conjunction with real hardware, evaluation boards and in-circuit emulators ensure that projects developed to exploit the power of the H8 / 300H family get to market quickly.

H8 / 300H CPU

The H8/300H CPU core has been developed as a 16 / 32-bit general purpose register machine. The H8/300H architecture maintains the high degree of High Level Language programming efficiency of the earlier H8 families. Furthermore, it provides a very effective level of performance by virtue of increased clock rates and an enhanced instruction set.

The CPU Model

The H8/300H CPU is a general purpose register machine as shown in *Figure 1*. The CPU comprises eight 32-bit registers, each being further dividable into 16 and 8-bit registers. Being general purpose there is no restriction placed on how each register is used, thus they can be used for pointer or data operations. The architecture also allows any of the data addressing modes to be used in conjunction with any register.

The registers are grouped into general purpose registers (called R0 to R7) and extended registers (called E0 to E7). The registers R0 to R7 are equivalent to those in the H8/300 CPU. Therefore, they can be used as eight 16-bit or sixteen 8-bit registers.

This new CPU core provides two modes of operation, H8/300 compatible or enhanced. In compatible mode the code developed for the 8-bit H8/300 family can be used without modification, allowing users of the H8/300 range to easily exploit the advanced peripheral set provided by the H8/300H. Enhanced mode allows the device to operate using its full 16 / 32-bit CPU core and extensive linear address space.

This rich set of general purpose registers provides the compiler writer with ample opportunities to optimise the code generated by the compiler. Local variables can be optimised into registers wherever possible, thus reducing the number of bytes of code needed to manipulate them. Also as each register can be used as an accumulator, index register or address pointer, the address arithmetic which must be performed by the compiler can be done very effectively.

In fact, the compiler can calculate the address of a variable in a register and in the next instruction use that same register as an address pointer to access the variable. In addition, register ER7 is used as a stack pointer, so all accesses to stack based data, an operation performed many times by HLLs can be performed very efficiently.

These features combine to significantly reduce the amount of code and time that is required to execute lines of C source code when compared to traditional

architectures which have limited numbers of fixed function accumulators and index registers.

The 32-bit register set also proves to be very useful in addressing large areas of memory, as a 24-bit pointer (this size pointer allows access to anywhere in the 16MBytes address space) can be stored and manipulated in one register.

In addition to the general purpose registers there are two control registers, a Condition Code Register (CCR) and a Program Counter (PC). The CCR is an 8-bit wide register which contains all the CPU flags such as overflow, zero and carry as well as the interrupt flags. The carry flag also doubles as a bit accumulator when the bit manipulation operations are used.

Operating Modes

The H8/300H CPU can operate in two basic modes, normal and advanced mode. The selection of operating mode is defined by the state of mode pins at reset.

The Normal Mode

In the normal mode the H8/300H CPU has access to a 64KBytes memory space. The exception vector table and stack have the same structure as for the H8/300 CPU. The extended registers E0 to E7 can be used as additional 16-bit data registers or combined with R0 to R7 as 32-bit general purpose registers. All instructions can be used, but for address calculations only the lower 16-bits of a general purpose register are significant.

The Advanced Mode

The full 16MBytes linear address space is available in the advanced mode. The exception vector table is organised into 32-bit units for every exception vector, with the valid vector address stored in the lower 24-bits of every unit. When a subroutine is called or during exception handling 4bytes are stacked (plus the CCR for exception handling) with the return PC value stored in the lower 24-bits.

Again in this mode the extended registers E0 to E7 can be used as additional 16-bit data registers or combined with R0 to R7 as 32-bit general purpose registers. If a 24-bit address pointer is used in a register, the upper 8-bits are ignored.

16MBytes Linear Address Space

To support large memory systems the linear address space of the H8/300H CPU core allows direct access to every memory location in the whole 16MBytes address space via 24-bit address pointers. The linear address space means there is no need to set up page registers and there are also no limitations on the size of code modules or data arrays and structures.

By combining both sets of 16-bit registers, the CPU core provides the eight 32-bit registers which can be used as operands for the 32-bit operations supported or as pointer registers into the linear address space. Depending on the application and device, the user can adapt the operation mode of the device to meet the system requirements. This includes the choice between different operating modes with 16MBytes, 1MBytes and 64KBytes address spaces.

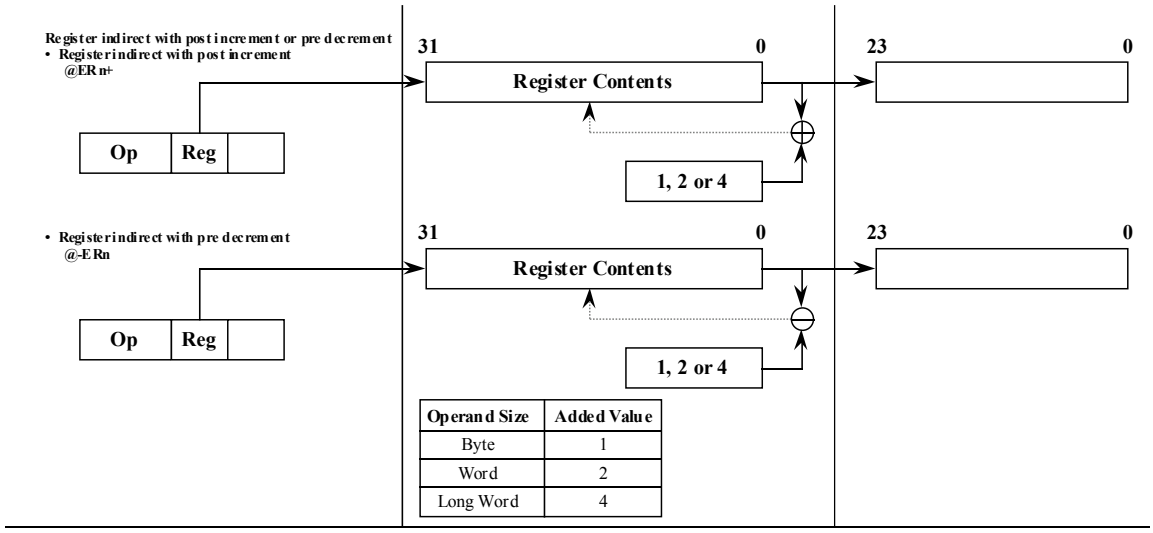
Addressing Modes

Another way a CPU architecture can support the efficiency of the compiler is by providing a full set of powerful and flexible addressing modes. A CPU which only provides rudimentary addressing modes makes a compiler inefficient in the

Figure 2 - H8 / 300H Addressing Modes

(1)	Register direct	R n
(2)	Register indirect	@ E R n
(3)	Register indirect with displacement	@ (d : 16, E R n) @ (d : 24, E R n)
(4)	Register indirect with post-increment / Register indirect with pre-decrement	@ E R n + @ - E R n
(5)	Absolute address	@ a a : 8 @ a a : 16 @ a a : 24
(6)	Immediate	# x x : 8 # x x : 16 # x x : 32
(7)	PC-relative	@ (d : 8, P C) @ (d : 16, P C)
(8)	Memory indrect	@ @ a a : 8

Figure 3 - Register Indirect with Post Increment & Pre Decrement



access of variables, thereby increasing both code size and execution time.

To ensure that the compiler is as efficient as possible the H8/300H CPU provides eight addressing modes as shown in *Figure 2*.

Each instruction can use a subset of the available addressing modes. The data transfer instructions can make use of all addressing modes except PC relative and memory indirect. All arithmetic and logical operations can use the register direct and immediate modes and the bit manipulation instructions use the register direct, register indirect and absolute addressing modes.

Supporting both array and stack data types, the H8/300H has indirect addressing with either post increment or pre decrement. These modes support byte, word and long word data (±1, 2 and 4) as shown in *Figure 3*.

Three absolute addressing modes are provided using 8, 16 or 24-bit absolute addresses. Using the 24-bit address the entire 16MBytes address space is accessible. The 8 and 16-bit absolute address modes assume that the upper byte or word of the address is H'FFFF or H'FF respectively. This allows for the efficient address specification for the on-chip I/O area and RAM areas which are both placed at the top of the address map. These shortened addresses save significant amounts of code when these areas are accessed.

Instruction Set

The H8/300H has a streamlined instruction set which suits the combined needs of HLL programming and embedded applications. It comprises of 62 instructions, with an emphasis on arithmetic instructions, address manipulation and bit processing. More than half of all instructions have an instruction length of only 2Bytes making very compact code. The full H8/300H instruction set is shown in *Figure 4*.

In comparison with the H8/300 CPU most of the data transfer, logical, shift and arithmetic instructions are improved to handle 16 and 32-bit data. New instructions added to the H8/300H include signed multiplication, sign extension, 16-bit branch instructions and a software trap instruction. *Figure 5* illustrates the

Figure 4 - H8 / 300H Instruction Set

Function	Instruction	Types
Data transfer	MOV, PUSH, POP, MOVTPE, MOVFPE	3
Arithmetic operations	ADD, SUB, ADDX, SUBX, INC, DEC, ADDS, SUBS, DAA, DAS, MULXU, DIVXU, MULXS, DIVXS, CMP, NEG, EXTS, EXTU	18
Logic operations	AND, OR, XOR, NOT	4
Shift operations	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BAND, BIAND, BOR, BIOR, BXOR, BIXOR, BLD, BILD, BST, BIST	14
Branch	Bcc, JMP, BSR, JSR, RTS	5
System control	TRAPA, RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	9
Block data transfer	EEPMOV	1
Total 62 types		

Figure 5 - H8 / 300H Added / Improved Instructions

Instruction \ Data Size	B	W	LW
Data Transfer MOV EEPMOV (Block Transfer)	○ ○	○ ●	●
Arithmetic Operations ADD, SUB, CMP MUL U/S DIV U/S EXTS/U	○ ○ ○	○* ● ● ●	● ●
Logical Operations AND, OR, XOR, etc.	○	●	●
Shift and Rotate SHAL, SHAR, ROT, etc.	○	●	●
System Control PUSH, POP TRAPA	○	●	●
Branch Bcc d : 16		●	

○ H8 / 300 & H8 / 300H
● H8 / 300H
*: improved addressing mode on H8 / 300H

new and improved instructions provided by the H8/300H.

The overall performance of the CPU core is very high with the 16MHz part giving a Dhrystone MIPS value of 1.9.

Arithmetic Instructions

In order to perform complex algorithms such as digital filtering the H8/300H is equipped with powerful arithmetic instructions, including addition and subtraction on 32-bit data and multiplication of 16 x 16-bit data and division of 32 / 16-bit data. Multiplication and division are both available as signed and unsigned operations, which eliminate the need for time consuming library calls. *Figure 6* gives a comprehensive guide to the execution speed of various arithmetic instructions with a clock of 16MHz.

Bit Processing

In microcontroller applications it is often necessary to manipulate data on a bit by bit basis. A good example would be where an I/O pin needs to be set to switch on a lamp or a solenoid. To meet this demand the H8/300H has 14 separate bit processing instructions which allow the programmer to manipulate bit data very easily.

It is also possible to perform boolean algebra on bit data using the carry flag of the CCR register as a bit accumulator. In a microcontroller application it is often necessary to perform a branch depending on the values of two bit flags located in RAM or I/O ports. Using the boolean operations provided by the H8/300H the first bit can be loaded into the carry flag. Then a bitwise logical operation can be executed using the second bit. This sequence would then be followed by a branch depending on the value of the carry flag.

Another feature of the H8/300H bit processing capability is its ability to access bits indirectly, using the value from a general purpose register as a bit pointer. This mechanism is shown in *Figure 7* and is useful for scanning a byte for set or cleared bits.

Block Move Instruction

Another efficient instruction provided by the H8/300H CPU is the EEPMOV or block data transfer. This is useful when a table stored in ROM has to be transferred to RAM for manipulation. Block sizes up to 64KBytes can be transferred with a single instruction.

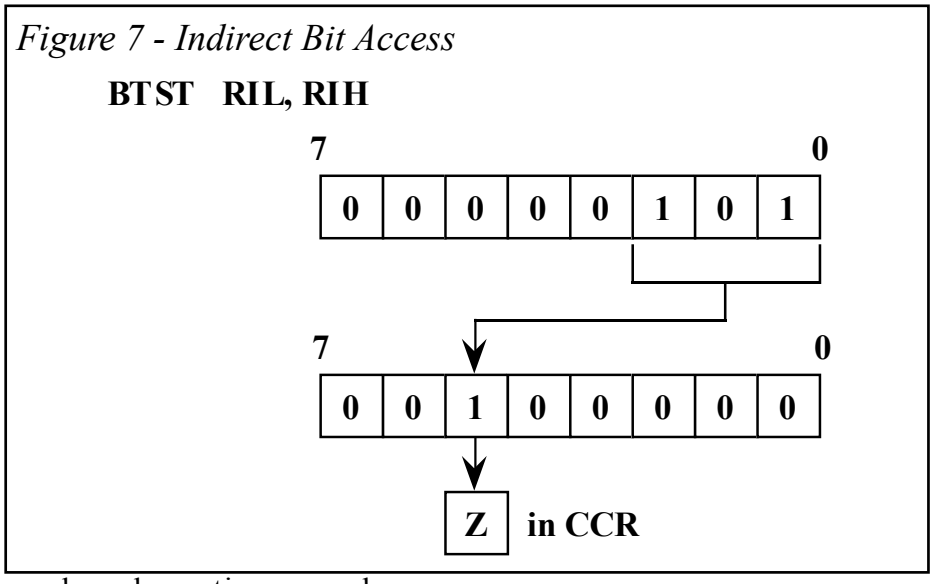
Software Interrupt

The TRAPA instruction has been added to the H8/300H CPU. This instructions implements a software interrupt, jumping to a service routine via one of four exception vectors (TRAPA 0 - 3). This operation can be used to implement fast, space efficient calls to often

Figure 6 - Performance of Arithmetic Operations (Ø = 16MHz)

ADD.L	**ERd, ERs**	**32-bit + 32-bit → 32-bit**	**125ns**
AND.L	**ERd, ERs**	**32-bit ^ 32-bit → 32-bit**	**125ns**
MULXS.W	**Rd, Rs**	**16-bit x 16-bit → 32-bit (signed)**	**1.5µs**
MULXU.W	**Rs**	**16-bit x 16-bit → 32-bit (unsigned)**	**1.375µs**
DIVXS	**ERd, Rs**	**32-bit ÷ 16-bit → 16-bit remainder / 16-bit quotient (signed)**	**1.5µs**
DIVXU	**ERd, Rs**	**32-bit ÷ 16-bit → 16-bit remainder / 16-bit quotient (unsigned)**	**1.375µs**

Figure 7 - Indirect Bit Access



used sub-routines such as schedulers and other O/S routines. The TRAPA instruction can also be used as a call to an error handling routine.

CPU States

The H8/300H CPU has four different processing states: program execution, exception handling, bus-released and power-down.

In the program execution state the CPU executes normal program instructions in sequence, while the exception handling state is a transient state in which the CPU executes an exception handling sequence in response to a reset, interrupt or other exception. In the bus-released state the external bus has been released to an external bus-master other than the CPU. In the power down mode the CPU is halted to conserve power.

The power down state includes three modes : sleep, software standby and hardware standby. These modes are supplemented in the H8/3048 range.

CPU Summary

The H8/300H CPU has been designed for systems which demand high arithmetic performance along with the ability to handle large data structures and program sizes efficiently. However, it retains operations important in microcontrollers such as bit processing. These attributes are combined with an architecture which ensures that HLL compilers can generate space and time efficient code.

Low Power Modes

The H8/300H Series has been designed to be a microcontroller with high performance and low power dissipation. It therefore can be used in 3V or 5V systems and only consumes 20mA (max) when operating at 3V and 8MHz.

To widen its use in battery operated equipment such as cellular telephones, an impressive set of low power modes are also provided on all devices.

Sleep Mode

In this mode the device switches off the clock to the CPU, but all of the on-board peripherals remain active, and register and memory contents are retained.

Sleep mode is entered via the "SLEEP" instruction; the CPU exits this mode whenever an enabled interrupt occurs.

A useful application for this mode is to reduce the average power consumed by a system, using a timer to"wake" the CPU after a period of slumber.

Once woken, the CPU can process for a period of time and then after loading the timer again the SLEEP instruction can be executed, again lowering the power consumption.

When sleep mode is entered the devices current consumption is reduced by approximately one third over its operating value.

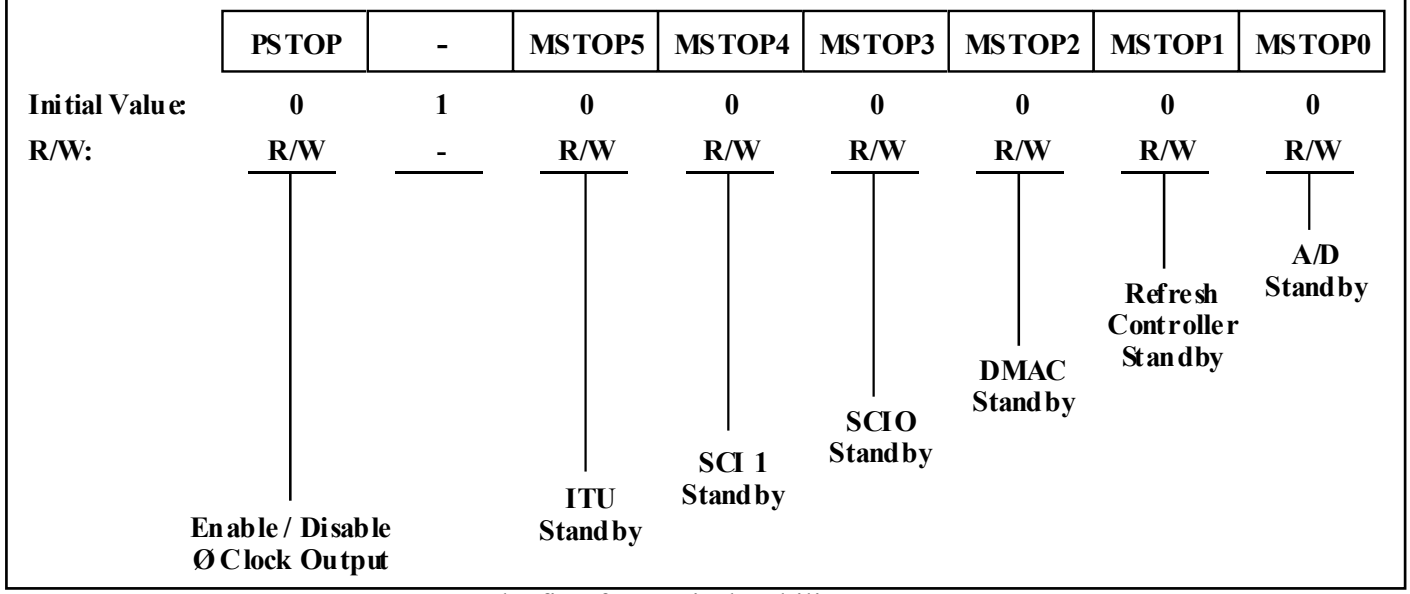
Software Standby Mode

Again, this mode can be entered using the SLEEP instruction, but in this case the on-chip oscillator is stopped completely, putting the device into software standby.

Standby current is very low with a maximum value of 5μA. This is coupled with a data retention voltage of 2V, allowing the microcontrollers internal RAM contents to be maintained using just two 1.5V battery cells or possibly a large "reservoir" capacitor.

During software standby mode, the microcontroller maintains the value

Figure 8 - H8 / 3048 Standby Control Register



of the I/O ports, so output ports can be set to the values the system requires during power down, with the knowledge that they will remain stable during software standby mode.

To exit from this mode, an external interrupt can be used, and a specialised timer circuit is provided to ensure that the on-chip oscillator has started and is stable before execution of the interrupt service routine begins.

Hardware Standby Mode

This mode allows the device to be put into the low power mode via an external pin. While in this mode the maximum current consumption is 5µA and to exit hardware standby the chip must be reset.

Extra Power Down Support - H8 / 3048

The H8/3048 range incorporates some extra power down options making it an ideal device in many high performance battery driven systems.

The first feature is the ability to put individual peripherals into standby mode via software. The control register used for this operation is shown in *Figure 8*.

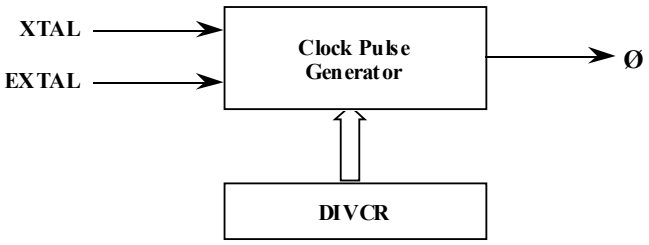
This feature makes the H8/3048 an ideal fit into digital cellular handsets as it mirrors the hardware designer's efforts to enable parts of the circuit to be powered down when they are not required.

Clock Gearing

Another power down feature provided by the H8/3048 is its ability to change operating frequency via a software command. This is achieved using a programmable clock divider which allows the clock to be divided by 1, 2, 4 or 8. The operation of this facility is shown in *Figure 9*.

Clock gearing allows the performance and power dissipation to be changed to suit the system's

Figure 9 - H8 / 3048 Clock Gearing



Bit 1□	Bit 0□	Divide Ratio□	Ø = 16MHz
DIV1□	DIV0		
0□	0□	1 / 1□	Ø = 16MHz
0□	1□	1 / 2□	Ø = 8MHz
1□	0□	1 / 4□	Ø = 4MHz
1□	1□	1 / 8□	Ø = 2MHz

Table 1 - Exception Vector Table

Exception Source	Vector Number	Vector Address *1
Reset	0	H'000 to H'0003
Reserved for system use	1	H'0004 to H'0007
	2	H'0008 to H'000B
	3	H'000C to H'000F
	4	H'0010 to H'0013
	5	H'0014 to H'0017
	6	H'0018 to H'001B
External Interrupt (NMI)	7	H'001C to H'001F
Trap Instruction (4 sources)	8	H'0020 to H'0023
	9	H'0024 to H'0027
	10	H'0028 to H'002B
	11	H'002C to H'002F
External Interrupt IRQ_0	12	H'0030 to H'0033
External Interrupt IRQ_1	13	H'0034 to H'0037
External Interrupt IRQ_2	14	H'0038 to H'003B
External Interrupt IRQ_3	15	H'003C to H'003F
External Interrupt IRQ_4	16	H'0040 to H'0043
External Interrupt IRQ_5	17	H'0044 to H'0047
External Interrupt IRQ_6	18	H'0048 to H'004B
External Interrupt IRQ_7	19	H'004C to H'004F
Internal Interrupts (On-chip peripherals)	20 to 60	H'0050 to H'0053 to H'00F0 to H'00F3

Note: *1 Lower 16-bits of the address

current mode. For example, when scanning a keyboard or other input device the divide by 8 option could be selected, but once a pressed key is detected the divide by 1 option can be selected to instantly give the device full speed operation.

Exceptions and Interrupts

Exceptions on the H8/300H CPU fit into four categories, reset (highest priority), external interrupts, internal interrupts and trap exceptions. The exception vector table for the H8/300H CPU core is shown in *Table 1*.

Trap Exceptions

When the TRAP instruction is executed, the program will start executing from the location specified in the relevant vector. The TRAP instruction has four vectors as specified by its argument. This exception can be used as an efficient mechanism for calling operating system functions, as it takes only 2Bytes to execute a TRAP operation, compared to 4Bytes for a BSR and 8Bytes for a JSR.

Interrupt Controller

The H8/300H interrupt controller (INTC) can be operated in two modes, either maintaining compatibility with the standard H8/300 INTC, or in a more advanced mode.

H8 / 300 Compatible Mode

In this mode, the acceptance of maskable interrupts is controlled by the I bit in the CCR. If I is set then all interrupts are disabled and if I is cleared then all interrupts are enabled. When an interrupt is accepted the INTC automatically sets the I bit, thus disabling any other maskable interrupt for the duration of the interrupt service routine (ISR), unless it is cleared by the user's code.

H8 / 300H Advanced Mode

To increase the power of the interrupt controller, an extra interrupt status bit is included in the condition code register, known as the UI or User Interrupt bit. This extended operation allows the user to specify raised priority interrupt sources, which are capable of interrupting a low priority ISR which is already running. The priority of individual interrupt sources is programmed in a number of interrupt priority registers (IPR).

The operation of the “UI” and “I” bits in the advanced mode is shown in *Table 2*. Now, when an ISR is running with the I bit set and the UI bit cleared, interrupts which have had their priority raised can interrupt and take over execution.

Table 2 - UE, I & UI Settings & Interrupt Handling

SYSCR	CCR		
UE	I	UI	Description
1	0	-	All interrupts are accepted. Interrupts with priority level 1 have higher priority
	1	-	No interrupts are accepted except NMI
0	0	-	All interrupts are accepted. Interrups with priority level 1 have higher priority
	1	0	NMI and interrupts with priority level 1 are accepted
		1	No interrupts are accepted except NMI

Table 3 - Interrupt Response Time (Worst Case)

No	Item	On-Chip Memory	External Memory			
			8-Bit Bus		16-Bit Bus	
			2 States	3 States	2 States	3 States
1	Interrupt priority decision	2	2 *1	2 *1	2 *1	2 *1
2	Maximum number of states until end of current instruction	1 to 23	1 to 27	1 to 31 *4	1 to 23	1 to 25 *4
3	Saving PC and CCR to stack	4	4	12 *4	4	6 *4
4	Vector fetch	4	8	12 *4	4	6 *4
5	Instruction prefetch *2	4	8	12 *4	4	6 *4
6	Internal processing *3	4	4	4	4	4
Total	(Clocks)	19 to 41	27 to 53	43 to 73	19 to 41	25 to 49

Notes: 1. One state for internal interrupts
2. Prefetch after the interrupt is accepted and prefetch of the first instruction in the interrupt service routine
3. Internal processing after the interrupt is accepted and internal processing after prefetch
4. The number of states increases if wait states are inserted in external memory access

External Interrupts

All the H8/300H devices include several external interrupts, including one non maskable interrupt (NMI). NMI can be programmed to be activated on either the rising or falling edge and the standard external interrupts can be programmed to recognise either a low level or a low going edge. The *Table on pages 10 and 11* contrasts the number of external interrupt pins available on the H8/300H family members.

Interrupt Vectors

To speed up the processing of interrupts, every interrupt source has its own vector. For example, for each serial port there are separate vectors for transmit, receive and error interrupts. Therefore, the ISR does not need to poll a peripheral block to find the source of any interrupt.

Interrupt Response Time

Interrupts are responded to rapidly on the H8/300H, as can be seen in *Table 3*. When code and data are both located in the on-chip memory, then the ISR will be reached within 2.6 microseconds (worst case) at 16MHz. If this is coupled with the individual vector structure provided by the H8/300H, the ISR can be performing useful work very quickly indeed.

Bus State Controller (BSC)

As one of the key reasons for using the H8/300H is its 16MBytes linear address space, it is likely that it will be used in a system with a large quantity of memory. Consequently, the H8/300H CPU core is supported by a powerful bus state controller (BSC) that allows the memory to be configured in the system in the most appropriate way.

Table 4 - Bus Controller Interfaces

	Bus Width (Bits)	Access Cycles	Size (Max)
SRAM	8 / 16	2 ~ 6	16MBytes
EPROM	8 / 16	2 ~ 6	16MBytes
PSRAM	8 / 16	2 ~ 6	16MBytes
DRAM	16	3 ~ 6	2MBytes

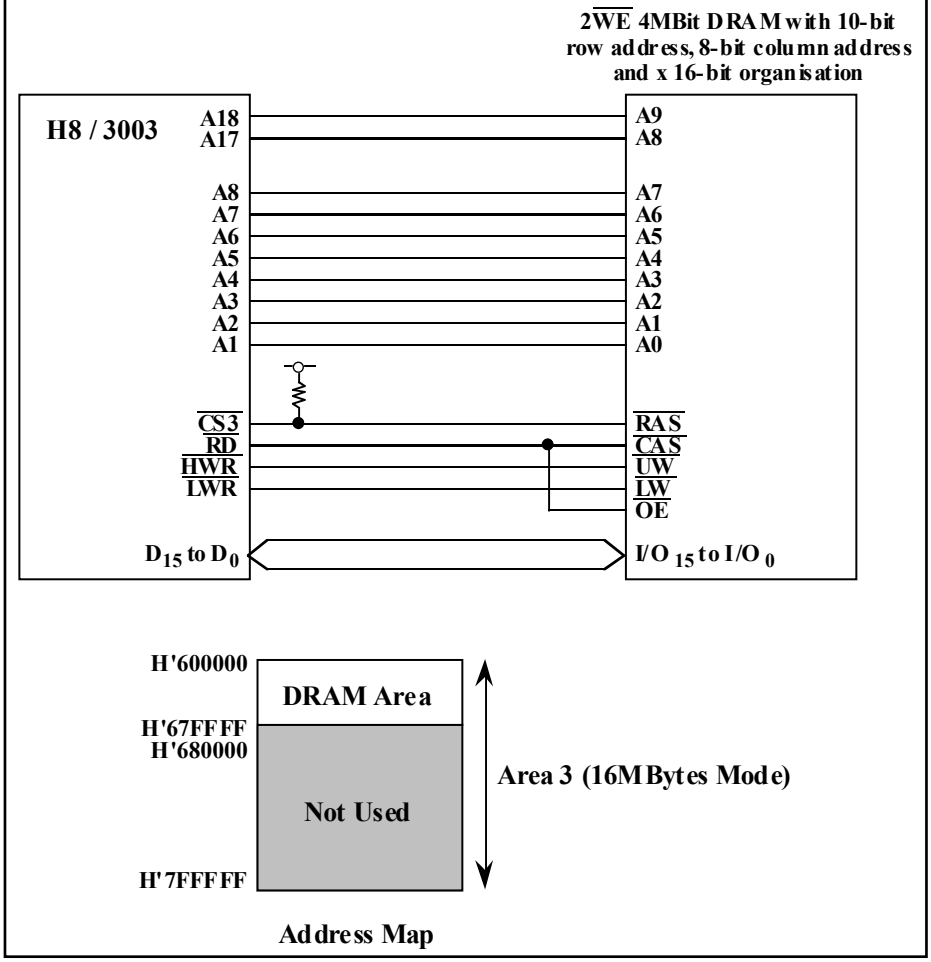
The H8/300H BSC is able to configure eight memory areas with their own independent attributes. In the advanced modes, these areas are either 256KBytes (1MBytes mode) or 2MBytes (16MBytes mode) in size. The attributes which can be set for each area are the bus width, the number of cycles for each external access and the wait state mode used. *Table 4* summarises the possible attributes of the memory which can be connected to the H8/300H.

When a memory area is initialised into the 3-state access cycle mode, the wait state controller can be activated for this area to allow slow

Table 5 - Wait State Controller Modes

Wait State Controller Mode	Operation
Pin Wait Mode 0	Tw wait states are inserted whenever the WAIT pin is sampled low. The wait state controller is otherwise disabled for memory areas so specified
Pin Wait Mode 1	The number of wait states programmed in the WSC are inserted and then the WAIT pin is sampled. If it is low then further wait states are inserted
Pin Auto-Wait Mode	If the WAIT pin is low when sampled the number of wait states programmed in the WSC are inserted and then the bus cycle is terminated
Programmable Wait Mode	For every access to the specified 3-state access area, the number of wait states programmed in the WSC are inserted automatically

Figure 11 - Interconnections & Address Map for 4M Bit DRAM (2WE)



devices to be connected to the bus of the H8/300H.

There are four wait modes available: the programmable wait mode, pin auto wait mode and the pin wait modes 0 and 1. The operation of these modes is summarised in *Table 5*.

Combining the BSC with the on-chip refresh controller, allows the H8 / 300H to be interfaced to a wide range of memory types, including DRAM, SRAM, PSRAM and EPROM with the minimum of glue logic. This function extends to the generation of chip selects corresponding to the memory area being accessed.

DRAM and PSRAM Interface

To provide a large area of RAM, the most cost effective memory type to use is DRAM. However, normally in an embedded system the external devices required to interface with DRAM often causes designers to use SRAM as the lowest cost system option.

To allow designers to utilise the full benefit of using DRAM in their system, the H8/300H has been equipped with a bus interface which can couple directly to DRAM with the minimum of external components.

This has been achieved by incorporating the logic required to produce the DRAM interface signals and a refresh controller into the BSC. For example, *Figure 11* shows the connection of a H8/3003 to a 2WE 4-Mbit DRAM.

The H8/300H supports 1Mbit and 4Mbit DRAM (2WE or 2CAS) in 16-bit wide configurations. The refresh controller, which is shown in *Figure 12*, can be programmed to produce a wide variety of refresh intervals, and the DRAM controller can put the DRAM into self refresh mode whenever the software standby mode is selected.

Pseudo Static RAM, which is another low cost memory technology, can also be interfaced to the H8/300H BSC, again using the refresh controller.

Figure 12 - Refresh Controller

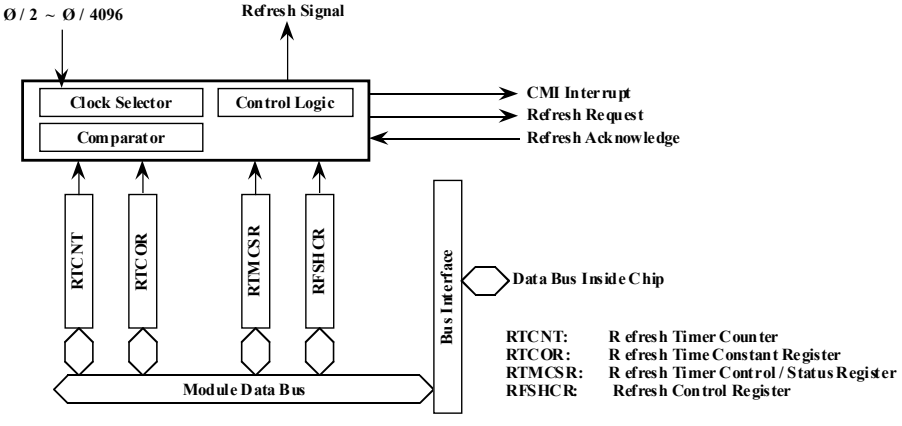
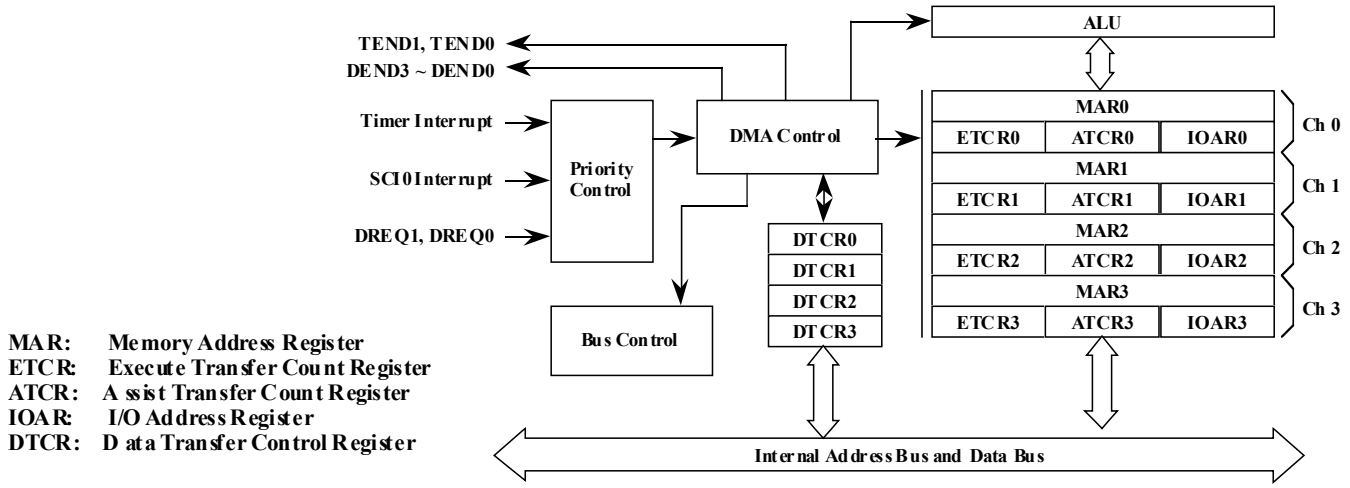


Figure 13 - DMA Controller



Chip Select Generation

The BSC on the H8/3003, H8/3002 and H8/304X devices can be used to generate chip select signals for different memory areas. These chip selects have the added benefit of becoming active at the same time as the address becomes valid, thus removing any decode delay.

Direct Memory Access Controller (DMAC)

To complement a CPU which provides high performance operation for complex algorithms and an address space which can handle large data structures and program modules, the addition of a direct memory access controller (DMAC) on-chip will significantly increase the performance of the system. The DMAC will allow the utilisation of the the whole CPU performance for the system's algorithms, while repetitive but important data transfer can be done via DMA.

Any external or internal interrupt that initiates a data transfer can be completely serviced by the DMA without any interrupts being handled by the CPU. This drastically reduces the CPU overhead needed for interrupt handling. When used with other on-chip peripherals such as the ITU, the DMAC allows the control of real time inputs and outputs, or it can be used to service the serial interfaces.

The basic H8/300H DMAC module incorporates four channels, as shown in *Figure 13*. However, the H8/3003 actually provides eight channels. Each channel in the DMAC module can be utilised to perform transfers between memory and I/O, while 2 channels need to be combined for memory to memory transfers. Byte and word transfer are possible in all available operating modes.

DMAC Modes

The DMAC provides a choice of short and full addressing modes, which allow the DMA to be adapted according to system requirements.

All of the DMAC modes are summarised in *Table 6.*

Short Address Mode

In this mode an 8-bit source address and 24-bit destination address (or vice versa) are used. During the transfer the 8-bit address (which

Table 6 - DMAC Functional Overview

	Transfer Mode	Activation	Address Reg. Length Source	Destination
Short address mode	I/O Mode •□Transfers 1Byte or 1 word per request •□Increments or decrements the memory address by 1 or 2 •□Executes 1 to 65,536 transfers	•□Compare match / input capture A interrupts from ITU channels 0 to 3 •□Transmit data empty interrupt from serial communication interface	24	8
	Idle Mode •□Transfers 1Byte or 1 word per request •□Holds the memory address fixed •□Executes 1 to 65,536 transfers	•□Receive data full interrupt from serial communication interface	8	24
	Repeat Mode •□Transfers 1Byte or 1 word per request •□Increments or decrements the memory address by 1 or 2 •□Executes a specified number (1 to 256) of transfers, then returns to the initial state and continues	•□External request	24	8
Full address mode	Normal Mode •□Auto request -□Retains the transfer request internally -□Executes a s	•□Auto request •□External request	24	24
	•□External request -□Transfers 1Byte or 1 word per request -□Execute:			
	Block Transfer •□Transfers 1 block of a specified size per request •□Executes 1 to 65,536 transfers •□Allows either the source or destination to be a fixed block area •□Block size can be 1 to 256 bytes or words	•□Compare match / input capture A interrupts from ITU channels 0 to 3 •□External request	24	24

Table 7 - ITU Functions

Item		Channel 0	Channel 1	Channel 2	Channel 3	Channel 4
Clock sources		Internal clocks: Ø, Ø / 2, Ø / 4, Ø / 8 External clocks: TCLKA, TCLKB, TCLKC, TCLKD, selectable independently				
General registers (output compare / input capture registers)		GRA0, GRB0	GRA1, GRB1	GRA2, GRB2	GRA3, GRB3	GRA4, GRB4
Buffer registers		-	-	-	B	-
Input / output pins		TIOCA0 TIOCB0	TIOCA1 TIOCB1	TIOCA2 TIOCB2	TIOCA3 TIOCB3	TIOCA4 TIOCB4
Output pins		-	-	-	-	TOCXA4 TOCSB4
Counter clearing function		GRA0 / GRB0 compare match or input capture	GRA1 / GRB1 compare match or input capture	GRA2 / GRB2 compare match or input capture	GRA3 / GRB3 compare match or input capture	GRA4 / GRB4 compare match or input capture
Compare match output	0	❍	❍	❍	❍	❍
	1	❍	❍	❍	❍	❍
	Toggle	❍	❍	-	❍	❍
Input capture function		❍	❍	❍	❍	❍
Synchronisation		❍	❍	❍	❍	❍
PWM mode		❍	❍	❍	❍	❍
Reset synchronised PWM mode		-	-	-	❍	❍
Complementary PWM mode		-	-	-	❍	❍
Phase counting mode		-	-	❍	-	-
Buffering		-	-	-	❍	❍
DMAC activation		GRA0 compare match or input capture Three sources • Compare match / input capture A0 • Compare match / input capture B0 • Overflow	GRA1 compare match or input capture Three sources • Compare match / input capture A1 • Compare match / input capture B1 • Overflow	GRA2 compare match or input capture Three sources • Compare match / input capture A2 • Compare match / input capture B2 • Overflow	GRA3 compare match or input capture Three sources • Compare match / input capture A3 • Compare match / input capture B3 • Overflow	- Three sources • Compare match / input capture A4 • Compare match / input capture B4 • Overflow

Legend: ❍ Available - Not Available

points to an I/O register) is fixed while the 24-bit address may change according to the way the channel has been initialised. In this mode DMA transfers are initiated by interrupts from the timer block, the serial port or via an external signal.

One byte or word of data is transferred per request and the 24-bit address incremented by one or two after each transfer. If a fixed memory address is required, then the channel can be put into an idle mode, where the 24-bit address will not increment. Up to 64K transfers can be performed before an interrupt is signalled to the CPU.

The DMA channel can also be set to automatically repeat up to 256 transfers continuously, with the DMA channel being reinitialised and restarted after the specified number of transfers has been performed. This is useful when cyclic data such as a control pattern for a stepper motor has to be transferred.

Full Address Mode

To perform memory to memory transfers the full address mode can be used. Here, the source and

Figure 14 - Integrated Timer Unit (ITU)

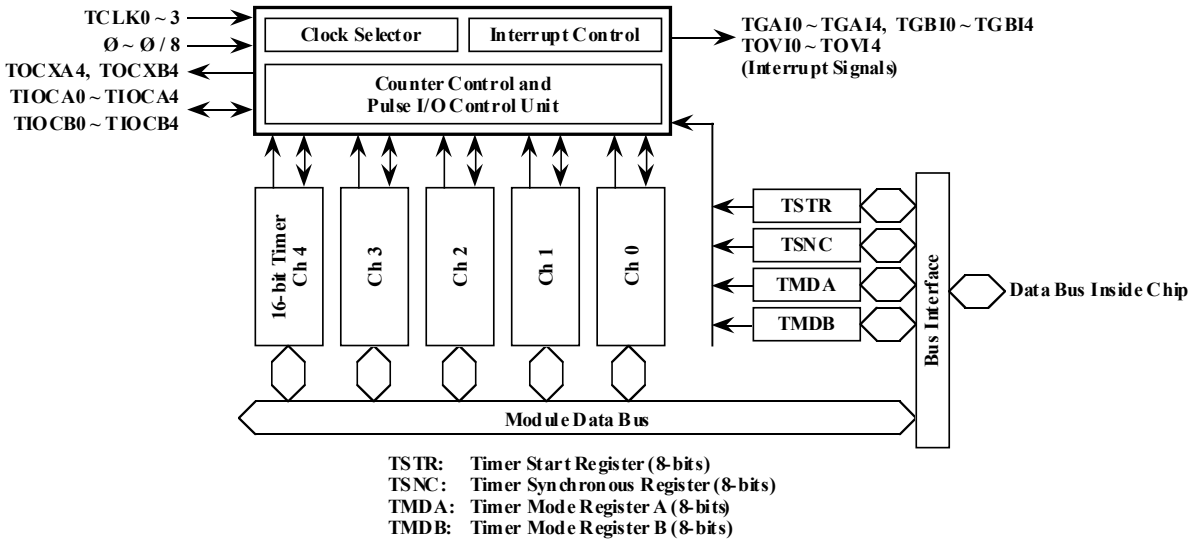
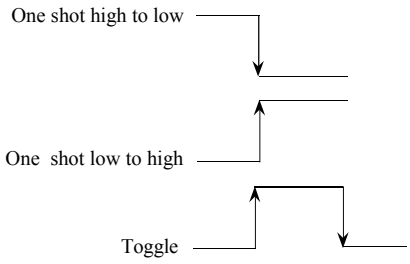


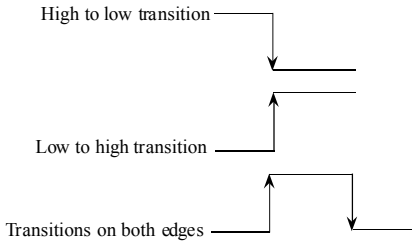
Figure 15 - Output Compare Waveforms



destination addresses are 24-bits wide each, and therefore memory to memory transfer can be performed between any areas of the full 16MBytes address space.

DMA transfers can be initiated by software command, external signals or interrupts from the timer block. This mode allows either a single transfer to occur per request, or for a block of data to be moved. In the block mode, the DMAC can be set up in a burst mode, taking over the bus from the processor until all the transfers are complete, or in a cycle steal mode where the processor and the DMAC share the bus.

Figure 17 - Input Capture Stimulas



DMAC Interrupts

The DMAC can be set up to provide an interrupt per request, or to only interrupt the processor when it has performed a programmed number of transfers.

Integrated Timer Unit (ITU)

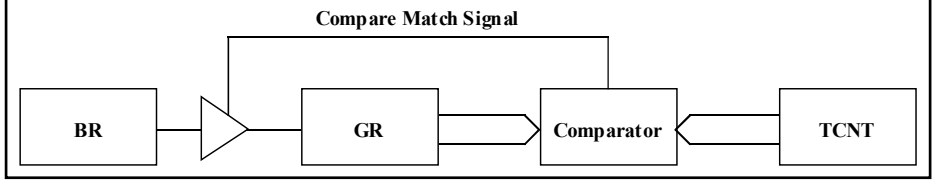
In many microcontroller based systems, very specialised timer functions are required. Often these timer functions are produced in a user developed ASIC device, because the timers provided by the microcontroller do not meet the performance required by the designer. The timer unit on the H8 / 300H, as shown in *Figure 14*, has been designed to allow maximum flexibility in its use, therefore allowing the designer to get the timer configuration required without resorting to an ASIC device.

a great deal of flexibility and reducing the need to develop specialised timer circuits externally to the microcontroller. This flexibility is further enhanced because each channel of timer can be set up individually, thus allowing the ITU block to perform several functions simultaneously. A full list of timer modes is shown in *Table 7*.

Output Compare Functions

To create output waveforms or

Figure 16 - Compare Match Buffering



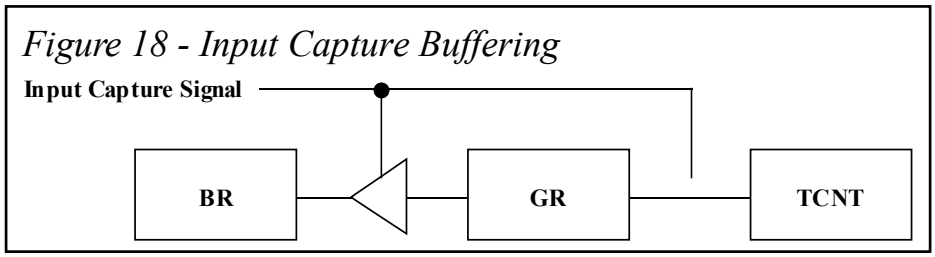
The ITU consists of five separate 16-bit timer channels, each of which can be clocked from an internal derivative of the system clock (Ø, Ø/2, Ø/4 and Ø/8) or from an external pin. If the Ø clock option is selected, then the minimum resolution of the timer is 62.5ns (Ø = 16MHz). The standard timer functions provided by the ITU include ten general registers (GR), which can be used as output compares or input captures. Thus the complete timer block provides up to ten pulse inputs or outputs. A further four 16-bit buffer registers (BR) reduce the overhead placed on the CPU when servicing the timer block.

The ITU can be used in a wide variety of modes, giving the designer

timed interrupts, the ITU provides up to 10 output compare registers. The output compares work by producing an output of a pre-programmed level and/or an interrupt when the value in the counter matches the value stored in one of the output compare registers. The events that can be initiated by these compare matches are transitions on an output pin as shown in *Figure 15*, a CPU interrupt (used for software timing functions), clearing the counter and the triggering of a DMA channel.

The buffer registers incorporated into timer channels 3 and 4 (BRA and BRB) can be used to buffer output events as shown in *Figure 16*. In this configuration a pulse

Figure 18 - Input Capture Buffering

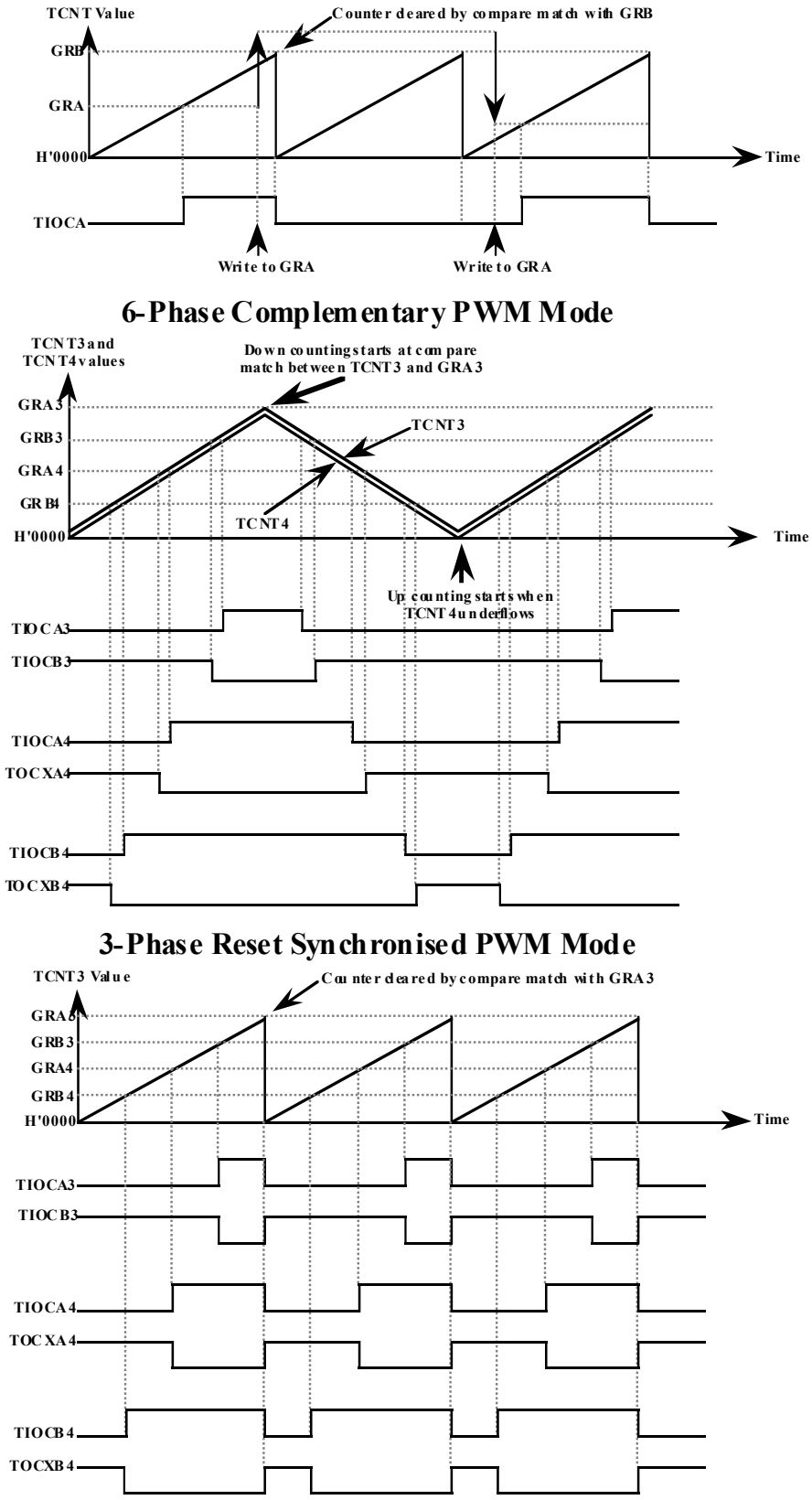


Feature	H8 / 3001	H8 / 3002	H8 / 3003	H8 / 303X	H8 / 3042	H8 / 3048
On-chip ROM / PROM	-	-	-	16K / 32K / 64K	32K / 48K / 64K	32K / 96K / 128K
On-chip RAM	512	512	512	512B / 1K / 2K	2K	2K / 4K
OTP Versions	-	-	-	✓	✓	✓
Flash Version	-	-	-	-	-	Avail: 1Q/95 (128K)
Address Range (max)						
Chip Selects	-	4	8	-	4	8
DMAC	-	4 channels	8 channels	-	4 channels	4 channels
16-bit Timers	5 channels	5 channels	5 channels	5 channels	5 channels	5 channels
16-bit Input Captures	10	10	10	10	10	10
16-bit Output Compares	10	10	10	10	10	10
Single Phase PWM	5	5	5	5	5	5
6-Phase PWM	✓	✓	✓	✓	✓	✓
Watchdog Timer	✗	✓	✓	✓	✓	✓
Analogue Inputs (10-bits)	4 channels	8 channels	8 channels	8 channels	8 channels	8 channels
Analogue Outputs (8-bits)	-	-	-	-	2 channels	2 channels
Serial Ports	1 channel	2 channels	2 channels	1 channel	2 channels	2 channels
Smart Card Interface	-	-	-	-	-	✓
Stepper Motor Outputs	16	16	16	16	16	16
I/O Ports (single chip mode)						
I/O Ports (16MByte mode)						
Clock Gearing Function	-	-	-	-	-	✓
Peripheral Standby Control	-	-	-	-	-	✓
Maximum Speed (5V)	16MHz	16MHz	16MHz	16MHz	16MHz	16MHz
Maximum Speed (3.3V)	13MHz					13MHz
Maximum Speed (2.7V)	8MHz	8MHz	8MHz	8MHz	8MHz	8MHz
Package	80-pin	100-pin	112-pin	80-pin	100-pin	100-pin
TQFP	✓		✓	✓	✓	✓

with duration down to one clock cycle (62.5ns at 16MHz) can be produced on the output pin.

As well as producing interrupts and events on output pins the compare function also allows DMA channels to be triggered to perform transfers. This function can used in conjunction with another peripheral block to control stepper motors. This function will be explained later.

Figure 19 - H8 / 300H PWM Modes



Input Capture Functions

The ITU provides up to 10 channels of input capture. In this mode the timer can be set up so that a transition on an input pin causes the value currently in the count register to be transferred into a capture register, thus time stamping that particular event. The ITU can be set to capture several types of transition, as shown in *Figure 17*. If required the timer unit can also clear the timer when the programmed external event occurs.

The two buffer registers (BRA and BRB) provided in timer channels 3 and 4 can be used to buffer input time stamps, as shown in *Figure 18*. This allows events which occur very close together to be time stamped using one capture pin. This feature can also be used to measure the width of an incoming pulse, by programming the capture input to be triggered on both the rising and falling edges.

By using input captures to measure the timing of external signals, very accurate measurements of variables such as frequency can be taken. The user can be sure that the accuracy of the measurement is not compromised by interrupt response time, as it is entirely a hardware driven facility.

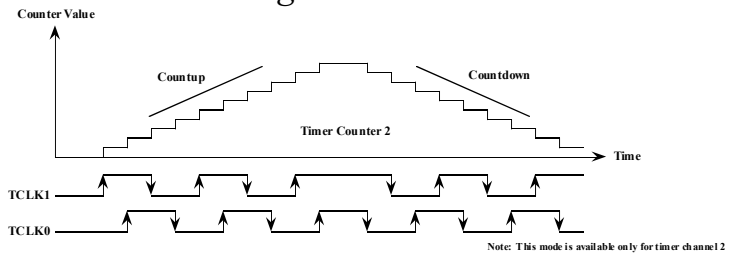
It is also possible to initiate DMA transfers when an input capture event occurs. This allows time stamps to be automatically placed

in memory via the DMA controller, without needing to interrupt the CPU.

Timer Synchronisation

To allow timer channels in the ITU to be used in synchronisation, it is possible to set up two or more timers so that they are simultaneously written to via software and cleared by compare matches or input captures. When timer channels are put into this mode then their input and output events are also synchronised.

Figure 20 Phase Counting Mode



Count Condition: Count All Edges

	Countdown				Countup			
TCLK0	↑	High	↓	Low	↑	Low	↓	High
TCLK1	Low	↑	High	↓	High	↑	Low	↓

PWM Operating Modes

The PWM (Pulse Width Modulation) modes of the ITU are described in *Figure 19*.

Standard PWM Mode

Each timer channel can be programmed to produce a single phase PWM output. Thus the ITU can output up to five separate channels of PWM. In this mode GRA controls the time when the pin goes high, and GRB when the pin goes low. Either GRA or GRB can be set to clear the counter, thus setting the frequency of the PWM output.

Table 8 - PWM Frequencies

PWM Resolution	16MHz	12MHz	10MHz	8MHz	6MHz
14-bit	976.5Hz	732.7Hz	610Hz	488Hz	365Hz
12-bit	3.9KHz	2.9KHz	2.4KHz	2KHz	1.5KHz
10-bit	15.6KHz	11.7KHz	9.7KHz	7.8KHz	5.8KHz
9-bit	31.2KHz	23.4KHz	19.5KHz	15.6KHz	11.7KHz
8-bit	62.4KHz	46.9KHz	39KHz	31.3KHz	23.4KHz
7-bit	124.8KHz	93.8KHz	78KHz	62.5KHz	46.8KHz

Table 8 shows the PWM frequencies which can be obtained against device clock speed and output resolution.

AC Motor Control Outputs

To provide the PWM signals required to drive AC machines, the ITU provides two further PWM modes: Complementary 6-phase PWM and Reset Synchronised 6-phase PWM. The main differences between these two modes are the transition points for the outputs and the provision of dead time between the phase outputs.

Complementary 6-Phase PWM

In this mode channels 3 and 4 are combined to produce three pairs of non-overlapping PWM waveforms, as described in *Figure 19*. As can be seen from this figure, in this mode TCNT3 and TCNT4 act as up/down counters, counting down from the point set by the compare match TCNT3 and GR3 and counting up from the point at which TCNT4 underflows.

The PWM waveforms are produced from compare matches with the general registers GRB3, GRA4 and GRB4. Using this mechanism, only three registers need to be reloaded

Figure 21 - TPC Block Diagram

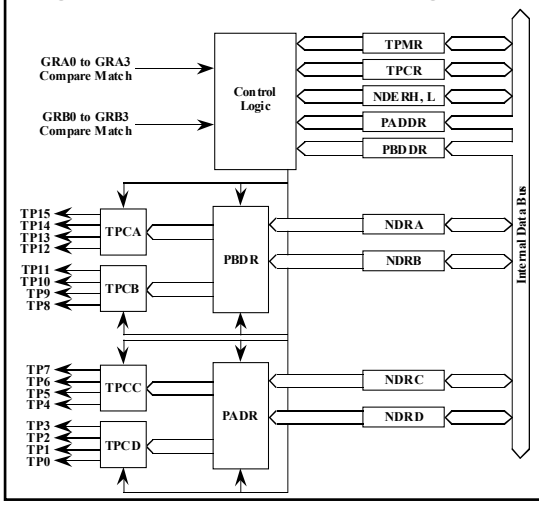
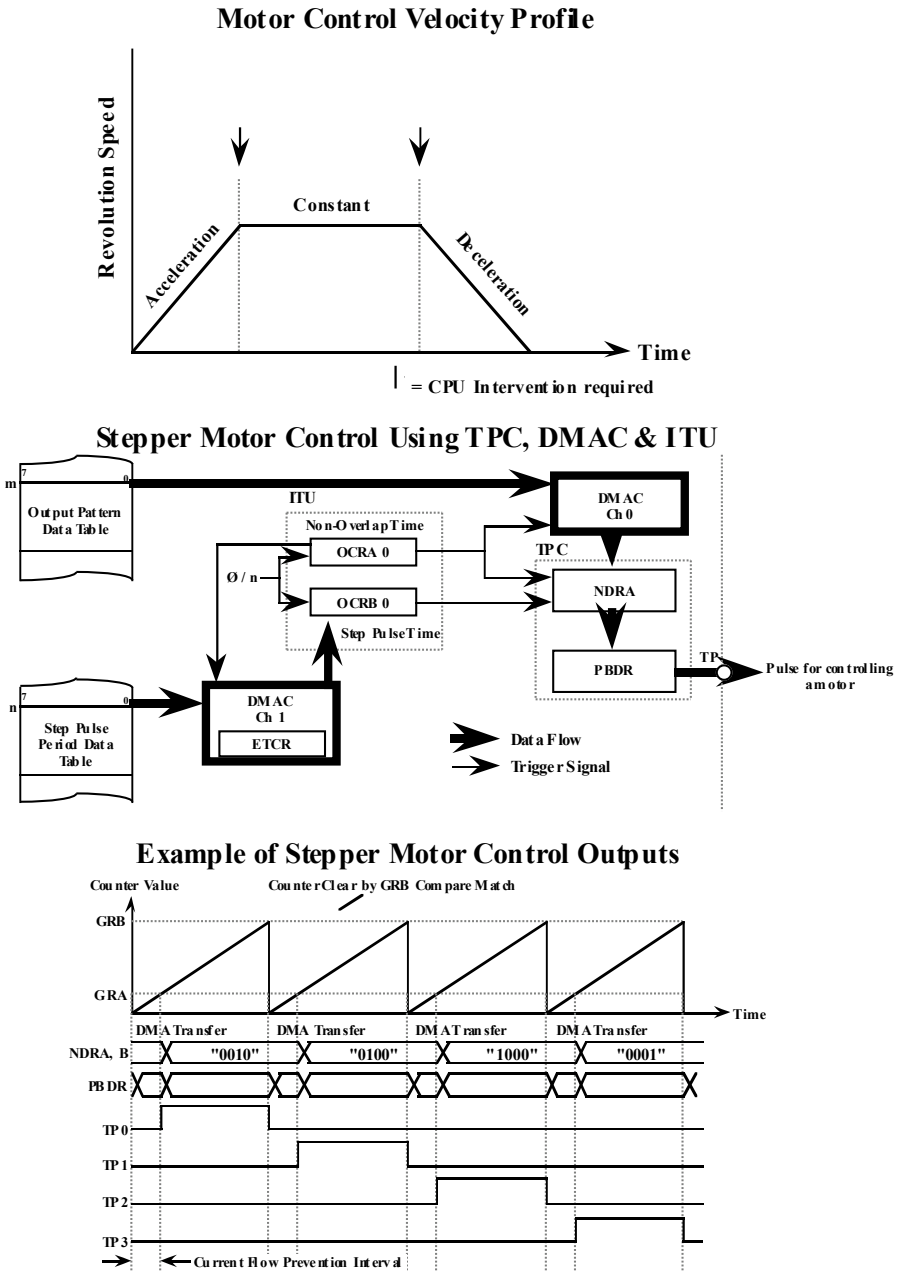


Figure 22 - Stepper Motor Control using the H8 / 300H



to change the modulation ratio, keeping the CPU overhead to a minimum.

In an AC motor control system, it is necessary to insert some deadtime between the switching of the complementary phases to ensure that no short circuit condition occurs through the two drivers. The ITU supports this function using the difference in value between the two timer channels used. Thus a totally programmable deadtime is supported in this mode.

Reset Synchronised PWM

This output mode is shown in *Figure 19* and it provides three pairs of complementary PWM waveforms, all having one common waveform transition point. In this mode TCNT3 counts up until it is cleared by a match with GRA3. The output pins toggle at compare matches between GRB3, GRA4, GRB4 and TCNT3 and they all toggle when TCNT3 is cleared.

Phase Counting Mode

This mode finds use in servo control systems, where the position and speed feedback comes from a 2-phase quadrature encoder. In this type of encoder the waveforms output change their phase relationship depending on the direction of motion, as shown in *Figure 20*.

By utilising the phase counting mode on the ITU, TCNT2 will count up or down depending on the phase of the incoming signals. Therefore, the value of TCNT2 will reflect the positional changes experienced by the encoder.

This facility removes the requirement for extra hardware or interrupt handlers for position monitoring. In this mode the comparators of channel 2 can also be used to generate interrupts, for example when a certain position is reached.

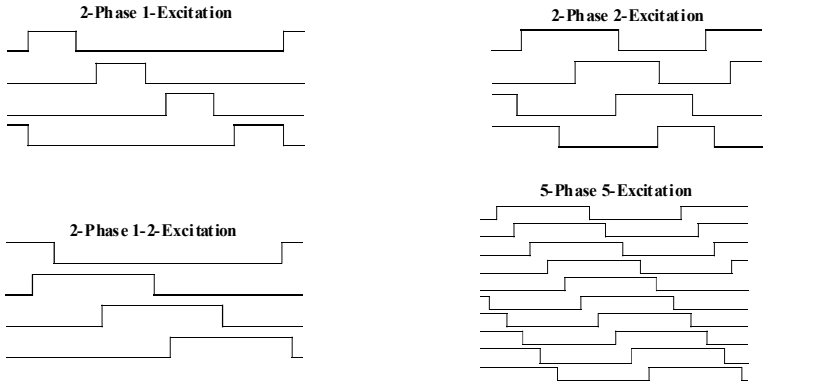
ITU Interrupts

The ITU can produce a total of 15 interrupts. This comprises 3 per channel, one for each general register and an overflow. In common with other H8/300H interrupts, each interrupt generated by the ITU has its own vector, thus reducing the amount of time needed to service any interrupt by removing the requirement for polling.

Programmable Timing Pattern Controller (TPC)

This peripheral can simultaneously output up to 16 waveforms, using a

Figure 23 - H8 / 3003, Examples of Stepper Motor Outputs and Resourced Needed



	2-Phase 1-Excitation	2-Phase 2-Excitation	2-Phase 1-2-Excitation	5-Phase	H8 / 3003 Total
DMAC	2 ch	2 ch	2 ch	2 ch	8 ch
TPC	4-bits	4-bits	4-bits	5-bits	16-bits
ITU	1 ch	1 ch	1 ch	1 ch	5 ch

strobe signal produced by the ITU. It is useful for generating the necessary waveforms for driving stepper motors. A block diagram of the TPC is shown in *Figure 21*.

The main registers of the TPC are the next data registers (NDRA~D). These registers are each 4-bits wide and contain the next data which will be transferred into the port data registers PADR and PBDR. This data is transferred using a strobe signal generated by the selected timer channel output compare. A separate timer channel can be specified to synchronise each group of 4-bits.

One of the main applications for this peripheral is the control of multiple stepper motors; if used in conjunction with the on-chip DMA controllers then this control can be performed with very little CPU supervision.

Stepper Motor Control with the TPC

A diagram showing how stepper motor control can be performed using the ITU, TPC and DMAC is shown in *Figure 22*. In this example a two phase stepper motor is being driven, using complementary transistors. It is therefore necessary to provide a dead time between the switching of the phases to eliminate any short circuit conditions between the high side and low side drivers.

Using compare matches from the ITU to stimulate the DMAC, new pattern data is provided to the TPC. This pattern data represents the next phase drive pattern required, and is stored in a memory table. The DMAC uses its memory to I/O function to transfer this data on each compare match. The TPC also uses the stimulus from the ITU to transfer the contents of the NDR to the port. Using a TPC mode where transitions on the port from 0 to 1 (ie switching on a phase) are only made on compare match A, a dead time, equal to the value in GRA, is inserted.

Figure 24 - Watchdog Internal Timer

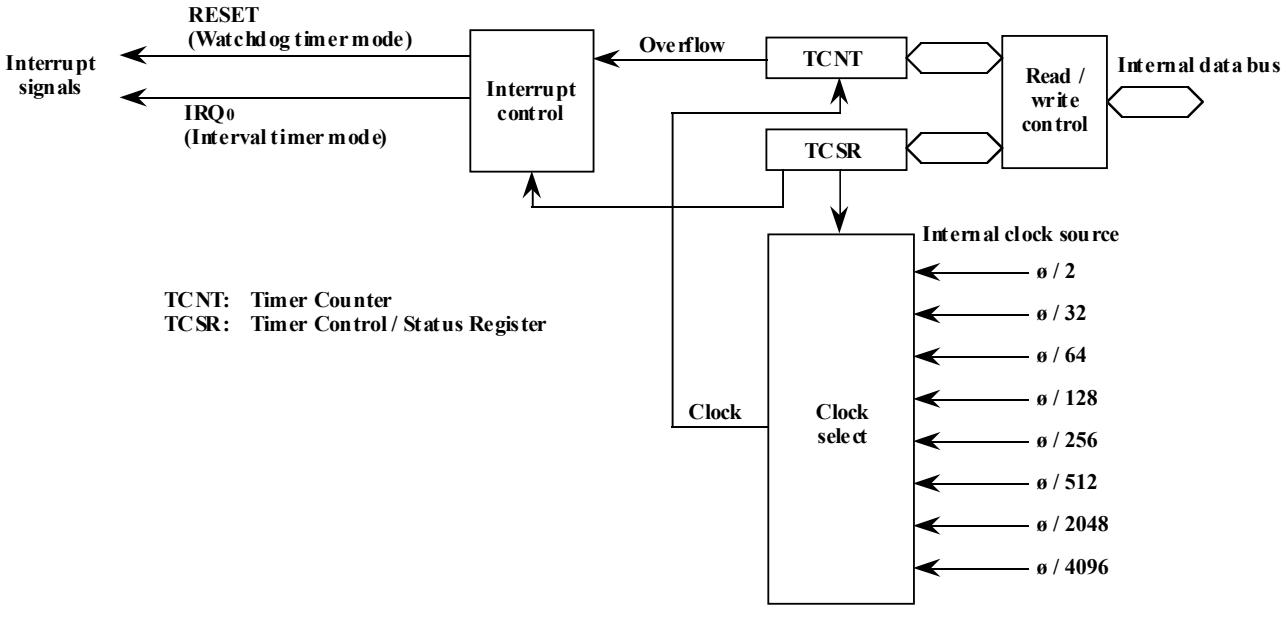
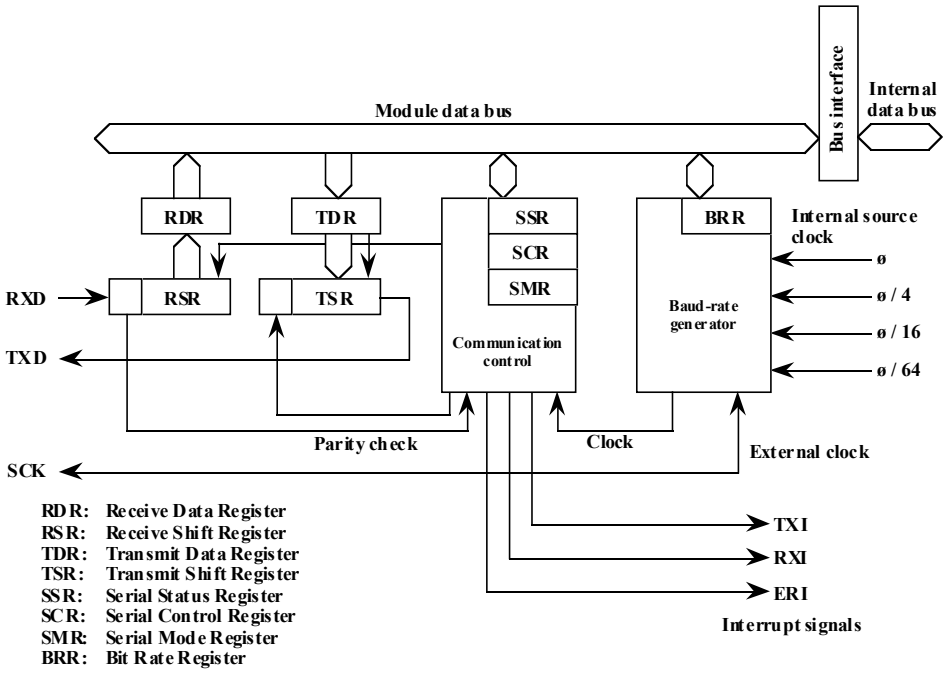


Figure 25 - Serial Communications Interface



The resulting output waveform is shown in *Figure 22*.

When controlling a stepper motor, providing the phase patterns onto the port pins is only part of the story. It is also necessary to modify the time between new patterns being output to allow acceleration and deceleration of the motor as shown by the velocity profile in *Figure 22*.

When the TPC, ITU and DMAC are working together, the acceleration and deceleration phases, as well as the steady speed phase can be controlled with minimal CPU overhead. The CPU need only get involved when a transition from one phase to another is made.

This is achieved by using a second memory to I/O DMA channel to reload the timer compare register after each new pattern has been output. Again the DMAC can take the next step period data from a table of values stored in the memory of the system. Therefore, by providing a table of increasing or decreasing values the motor can be decelerated or accelerated with no CPU intervention.

The flexibility of the H8/300H's stepper motor control functions will even allow multiple motors to be controlled simultaneously by one device. *Figure 23* shows examples of the stepper motors which can be controlled, and contrasts the resources required with the amount of resource available on the H8/3003.

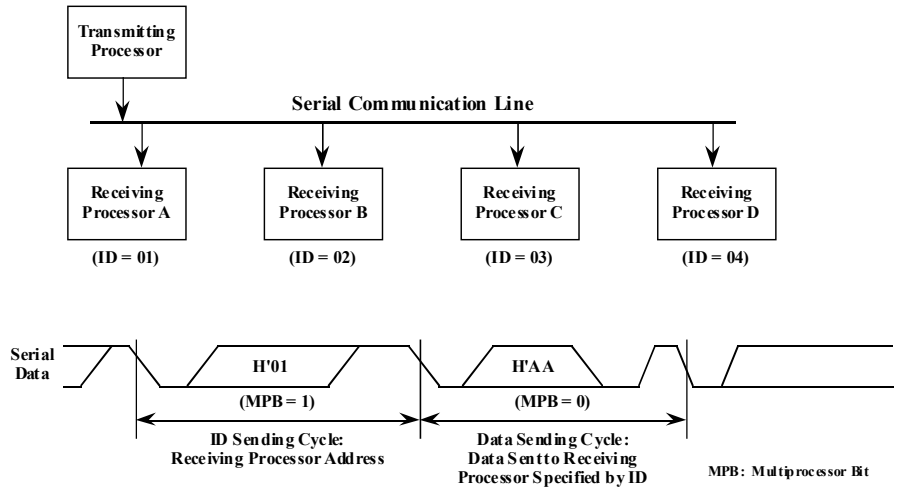
Watchdog Timer

Often, a watchdog timer is a very important feature in any embedded application. It is used to ensure that any"mishap" in the system(such as

Table 9 - BRR Settings for Typical Baud Rates (Asynchronous Mode) - Ø = 10MHz

Baud Rate (Baud)	Baud Rate Generator Input Clock	BRR Value	Error (%)
110	Ø / 16	174	-0.26
150	Ø / 16	127	0
300	Ø / 4	255	0
600	Ø / 4	127	0
1,200	Ø	255	0
2,400	Ø	127	0
4,800	Ø	63	0
9,600	Ø	31	0
19,200	Ø	15	0
31,250	Ø	9	-1.7
38,400	Ø	7	0

Figure 26 - Example of Communication Among Processors using Multiprocessor Format



a noise induced software crash) is rectified as quickly as possible.

The principle behind a watchdog timer is very simple - a counter is constantly counting upwards, and correctly operating software ensures that this counter never overflows by continuously resetting the count. If the software crashes and the counter overflows, the watchdog"barks" and sends some stimulus to the microcontroller (normally a reset) to restart system operations in a controlled manner.

All H8/300H devices are equipped with a timer, which can be used either as a watchdog or as an interval timer. Its "bark" is a reset if it is used as a watchdog. A diagram of the watchdog is shown in *Figure 24*.

Serial Communications

This form of communication has many uses in microcontroller applications, such as inter-device communications, diagnostics, host communication, and even as an interface to peripherals.

All H8/300H devices are equipped with at least one and very often two channels of serial communication interface (SCI). These channels can be used for either synchronous or asynchronous communications. This type of SCI is standard across the H8/300H range.

As shown in *Figure 25* each SCI channel has its own integral Baud rate generator, so many Baud rates can be produced from the microcontroller's internal clock, without using any other timers. *Table 9* shows a selection of the Baud rates available from this Baud rate generator. (Please note that this is only an example of the many Baud rates available from the H8/300H SCI - see the appropriate Hardware Manual for a complete list)

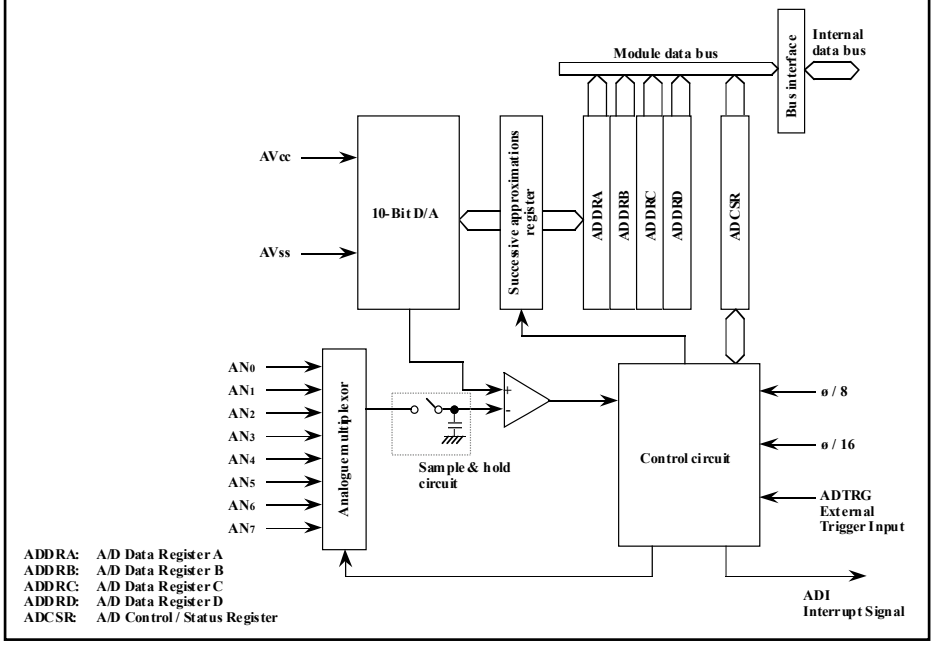
As well as using the integral Baud rate generator, each SCI can be configured to use an external serial clock.

To allow "back to back" transmission or reception of serial data, the SCI has double buffered transmit and receive shift registers.

The H8/300H serial ports also support multiprocessor communications using a master slave configuration in addition to the standard modes.

In this mode, communication between devices is performed using an additional multiprocessor bit (MPB) which is added to the data transmitted. This bit is used to differentitate between data frames and address frames. Thus, any

Figure 27 - Analogue / Digital Converter



frame sent from the master with the MPB set to one can be used to activate the required slave.

Slave devices on this network will only produce a receive interrupt when a frame is received with the MPB set, so the interrupt handler can check the address which has been transmitted.

The operation of this protocol is shown in *Figure 26*.

Receive data errors are trapped using three error conditions - overflow, framing and parity. These three errors are indicated via one interrupt vector and three status flags in the serial status register.

Analogue to Digital Converter

In many microcontroller based systems, some way of measuring real or analogue electrical values is necessary. With this in mind, all members of the H8/300H family are equipped with a 10-bit A/D convertor. The A/D convertor is illustrated in *Figure 27*.

The converter works using a successive approximation algorithm and conversions take 138 states (or 8.6µs if Ø - 16MHz).

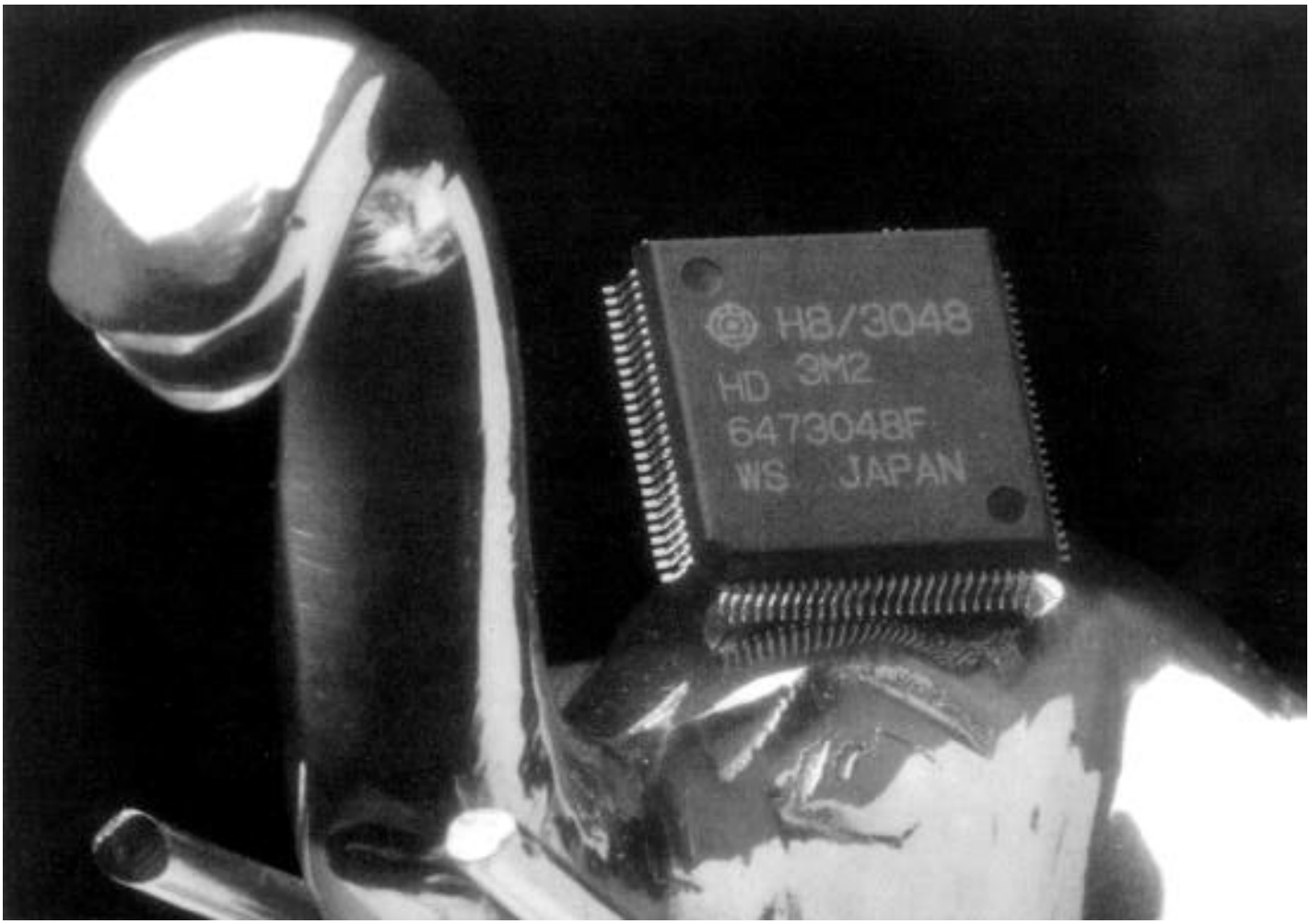
Up to 8 inputs can be converted; this is achieved using an 8 channel multiplexer between the input port and the A/D. A sample and hold capacitor is used to ensure that once a conversion begins, a change of the input value will not be reflected in a different conversion result.

As well as being able to perform single conversion(where only one channel is converted), the H8/300H A/D convertor can also be used to scan up to four channels. To support this mode of operation, four A/D result registers are provided.

Once the scan mode is selected, each channel specified is converted sequentially with the conversion value being stored in the appropriate result register. This mode of operation allows the user software to sample the current analogue value of an input by simply reading the appropriate data register.

Digital to Analogue Converter

The H8/304X device incorporates an 8-bit digital to analogue converter which has a maximum conversion time of 6.2µs (Ø = 16MHz). Two outputs are provided and multiplied with the analogue to



digital inputs. The output voltage range is from 0V through to the A/D reference voltage.

On-Board Memory

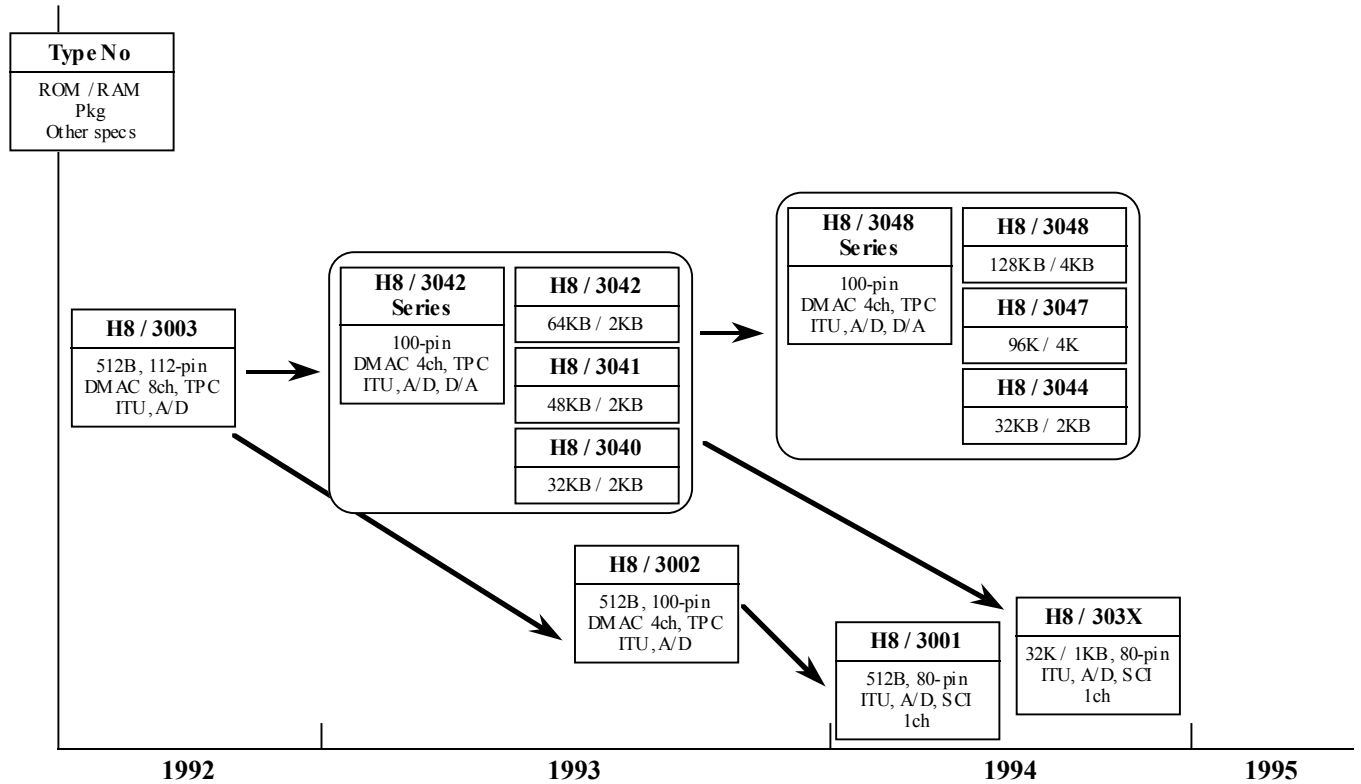
Often, due to its power and complexity, a H8/300H device will be used in applications where the program size is very large. At first glance, these large programs appear to preclude using any microcontroller in single chip mode.

But large programs place no restriction on the H8 / 300H family, as unparalleled sizes of on-chip program and data memory are available, even in its' smallest package.

For example the H8 / 3048 with 128K of PROM / ROM and 4K of RAM in a single chip, 100-pin device measuring just 17.2mm across its gull wings.

New variants of the H8 / 300H family are also under development. These include the H8 / 3048F with 128KBytes of sector eraseable Flash memory and 4KBytes of RAM.

H8 / 300H Series Evolution



H8 / 300H Part Numbers Defined

HD64 1 3002 VF 8

Hitachi Digital

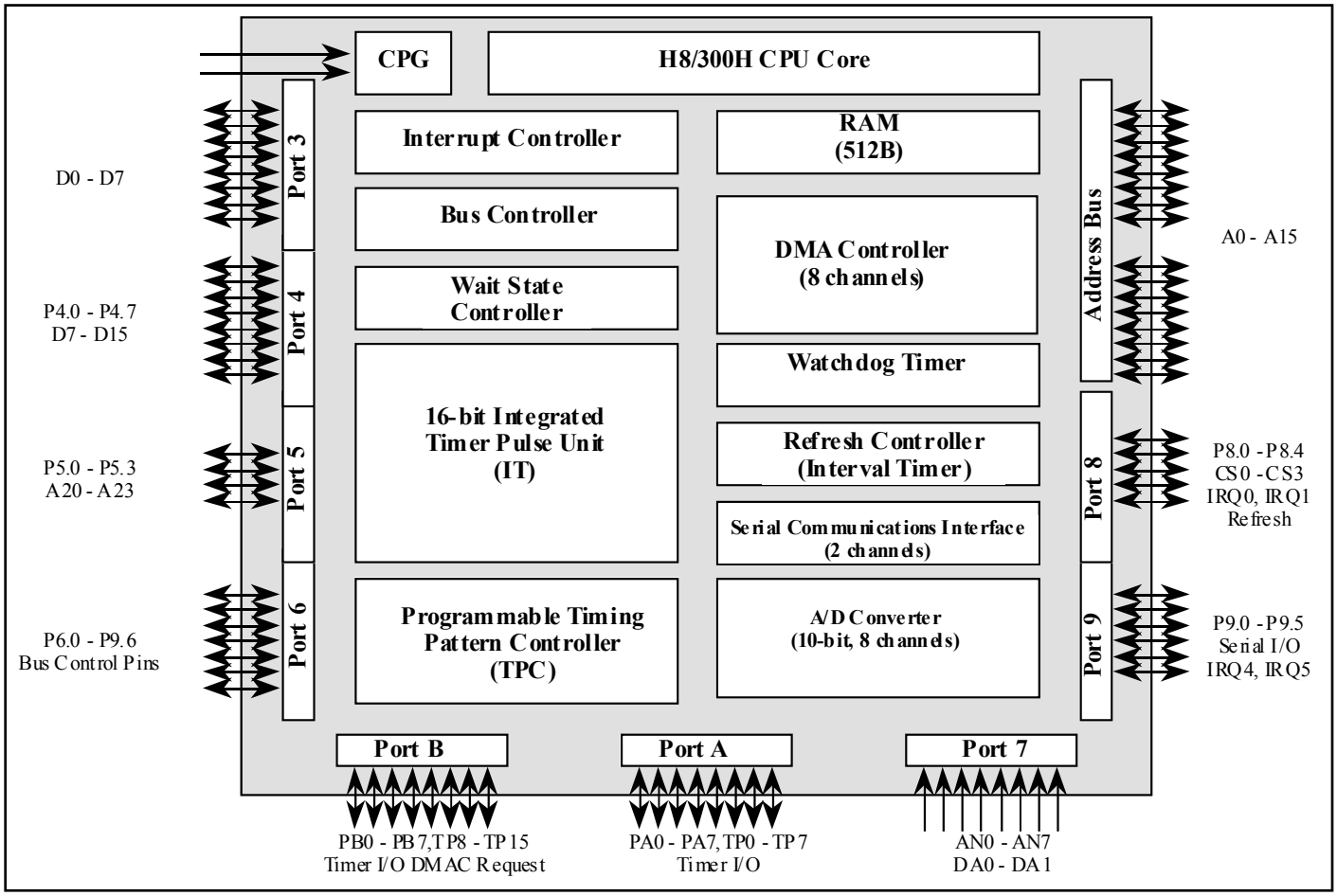
Memory Type
1 = ROMless
7 = ZTAT
3 = Mask ROM
F = Flash

Device Number
3001
3002
3003
303X
304X

Package / Voltage
F = FP (5V)
TF = TQFP (5V)
VF = FP (3V)
VX = TQFP (3V)

Speed
18MHz
16MHz
13MHz
12MHz
10MHz
8MHz

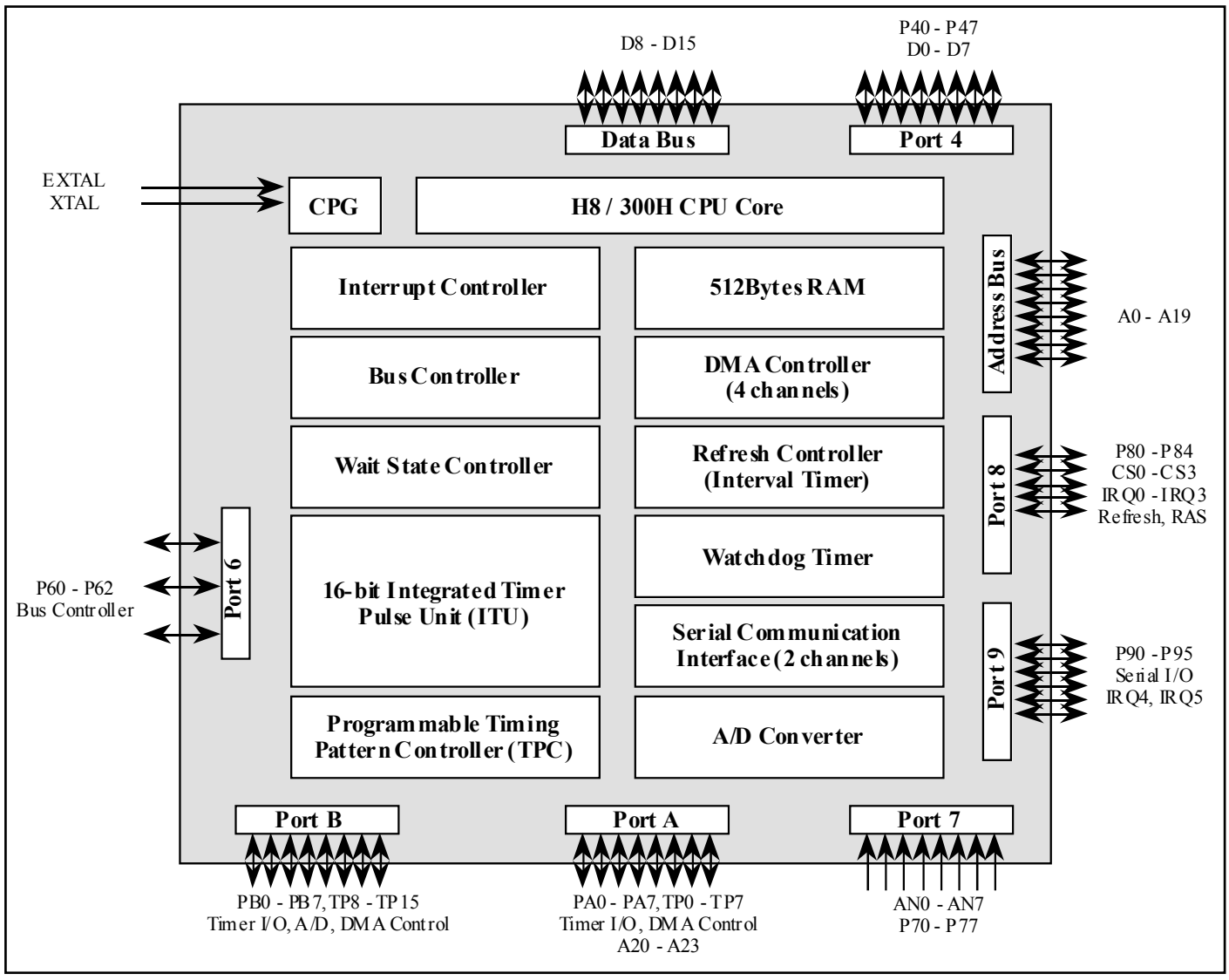
H8 / 3003



The H8/3003 has peripheral functions which are well suited to the needs of office automation applications. By coupling its 8 channel DMAC with the TPC up to 8 two phase stepper motors can be controlled independently of the CPU, making it ideal for even the most demanding printer system.

Part Number	Speed (MHz)	Package	Voltage	Comments
HD6413003F16	16, 12, 10	QFP-112	5V	1:2 clock
HD6413003TF16	16, 12, 10	QFP-112	5V	1:1 clock
HD6413003VF8	8	QFP-112	3V	1:2 clock
HD6413003VTF8	8	QFP-112	3V	1:1 clock

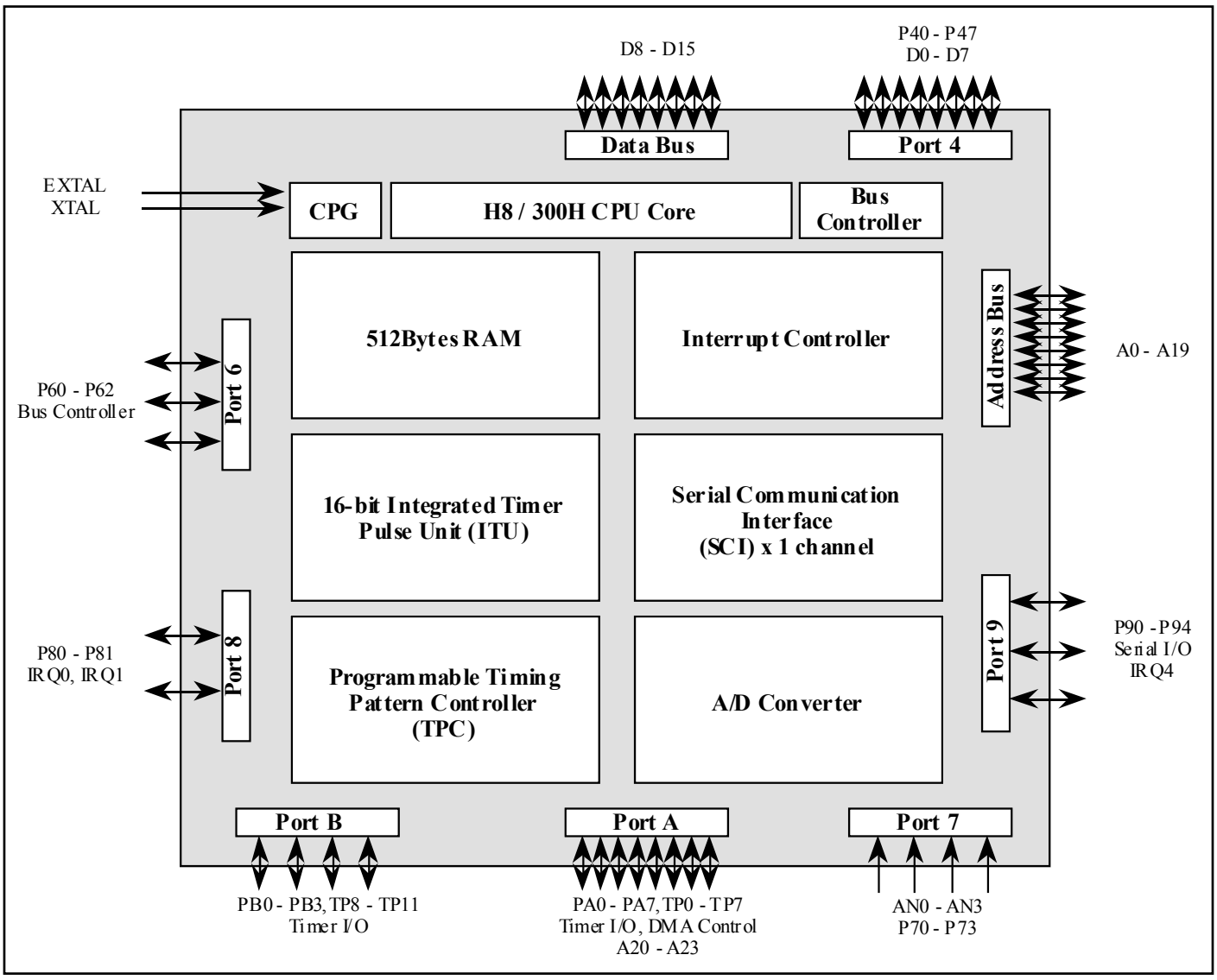
H8 / 3002



H8/3002 is a ROMless controller with the peripheral capacity to manage a complex system such as a cellular telephone or a portable ink jet printer. The H8/3002 can be supplied in a very space efficient thin QFP package, making it ideal for space limited systems.

Part Number	Speed (MHz)	Package	Voltage
HD6413002F	16, 12, 10	QFP-100	5V
HD6413002TF	16, 12, 10	TQFP-100	5V
HD6413002VF	8	QFP-100	3V
HD6413002VTF	8	TQFP-100	3V

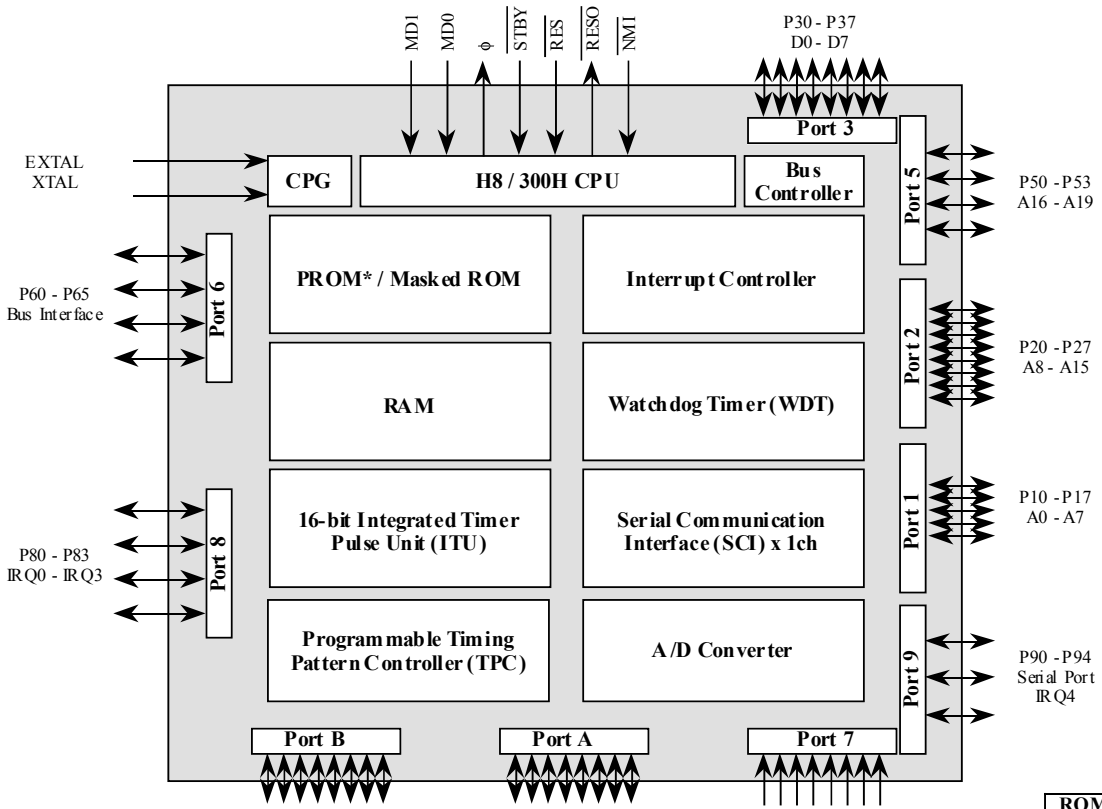
H8 / 3001



The H8/3001 is the smallest ROMless member of the H8/300H family. It incorporates a simplified peripheral set plus 512Bytes of RAM in an 80-pin QFP or TQFP package.

Part Number	Speed (MHz)	Package	Voltage
HD6413001F	16, 12, 10	QFP-80	5V
HD6413001TF	16, 12, 10	TQFP-80	5V
HD6413001VF	8	QFP-80	3V
HD6413001VTF	8	TQFP-80	3V

H8 / 3032

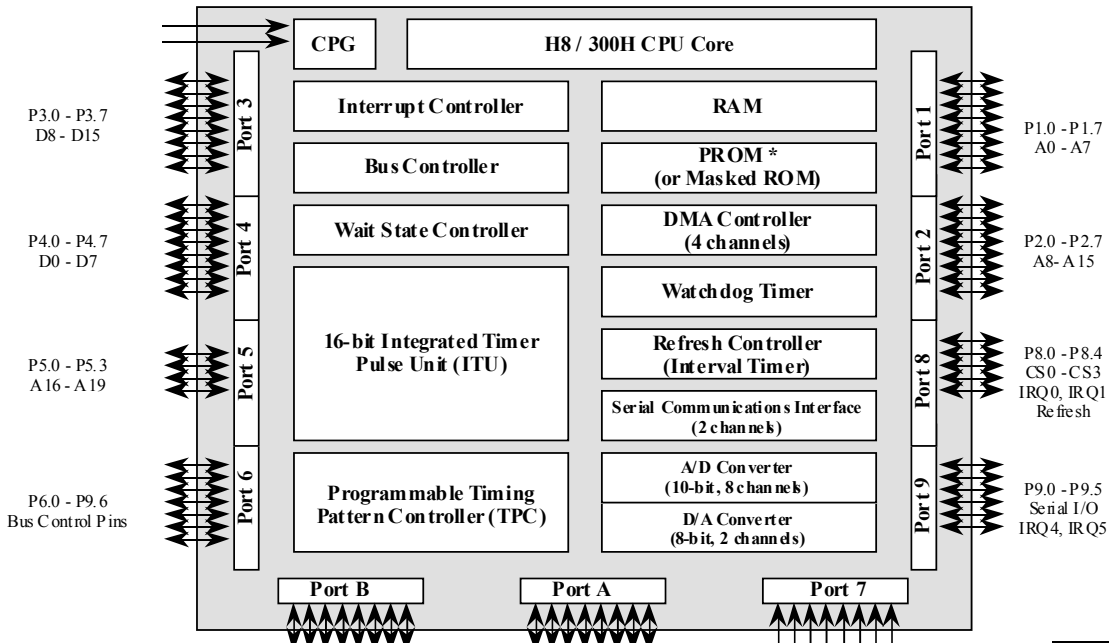


	ROM / OTP Size	RAM Size
H8/3032	64K	2K
H8/3031	32K	1K
H8/3030	16K	512B

The H8/3032 family is the smallest single chip version of the H8/300H range. It is packaged in an 80-pin QFP or TQFP package and offers a wide range of on-chip memory options. It is ideal for small scale embedded applications which need 16-bit performance.

Part Number	Speed (MHz)	Package	Voltage	ROM / RAM	OTP	Part Number	Speed (MHz)	Package	Voltage	ROM / RAM	OTP
HD6473032F	16, 12, 10	QFP-80	5V	64K / 2K	✓	HD6433031F	16, 12, 10	QFP-80	5V	32K / 1K	✗
HD6473032TF	16, 12, 11	TQFP-80	5V	64K / 2K	✓	HD6433031TF	16, 12, 10	TQFP-80	5V	32K / 1K	✗
HD6473032VF	8	QFP-80	3V	64K / 2K	✓	HD6433031VF	8	QFP-80	3V	32K / 1K	✗
HD6473032VTF	8	TQFP-80	3V	64K / 2K	✓	HD6433031VTF	8	TQFP-80	3V	32K / 1K	✗
HD6433032F	16, 12, 10	QFP-80	5V	64K / 2K	✗	HD6433030F	16, 12, 10	QFP-80	5V	16K / 512B	✗
HD6433032TF	16, 12, 10	TQFP-80	5V	64K / 2K	✗	HD6433030TF	16, 12, 10	TQFP-80	5V	16K / 512B	✗
HD6433032VF	8	QFP-80	3V	64K / 2K	✗	HD6433030VF	8	QFP-80	3V	16K / 512B	✗
HD6433032VTF	8	TQFP-80	3V	64K / 2K	✗	HD6433030VTF	8	TQFP-80	3V	16K / 512B	✗

H8 / 3042 Range

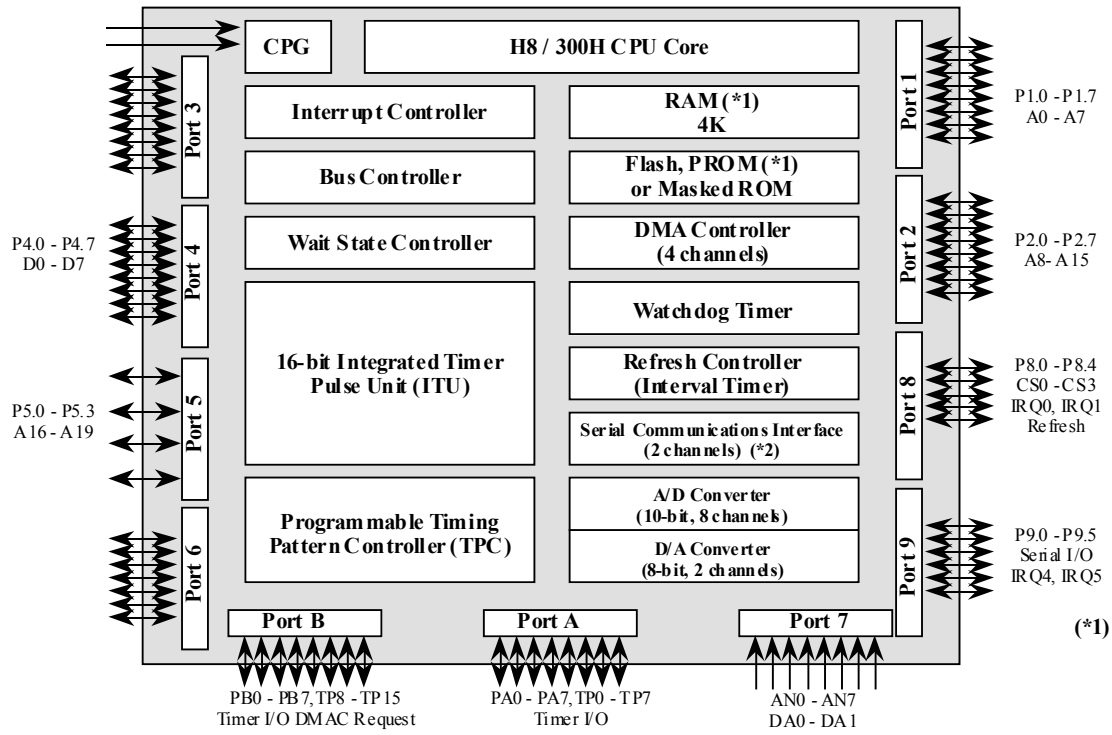


	ROM / OTP Size	RAM Size
H8/3042	64K	2K
H8/3041	48K	2K
H8/3040	32K	2K

The H8/3042 range offers several different program memory options, including 64K PROM / ROM, 48K ROM or 32K ROM. It also includes a large 2K RAM and a wealth of on-chip peripherals.

Part Number	Speed (MHz)	Package	Voltage	ROM Size	Part Number	Speed (MHz)	Package	Voltage	ROM Size
HD6473042F	16, 12, 10	QFP-100	5V	64K (OTP)	HD6433041F	16, 12, 10	QFP-100	5V	48K (ROM)
HD6473042TF	16, 12, 10	TQFP-100	5V	64K (OTP)	HD6433041TF	16, 12, 10	TQFP-100	5V	48K (ROM)
HD6473042VF	8	QFP-100	3V	64K (OTP)	HD6433041VF	8	QFP-100	3V	48K (ROM)
HD6473042VTF	8	TQFP-100	3V	64K (OTP)	HD6433041VTF	8	TQFP-100	3V	48K (ROM)
HD6433042F	16, 12, 10	QFP-100	5V	64K (ROM)	HD6433040F	16, 12, 10	QFP-100	5V	32K (ROM)
HD6433042TF	16, 12, 10	TQFP-100	5V	64K (ROM)	HD6433040TF	16, 12, 10	TQFP-100	5V	32K (ROM)
HD6433042VF	8	QFP-100	3V	64K (ROM)	HD6433040VF	8	QFP-100	3V	32K (ROM)
HD6433042VTF	8	TQFP-100	3V	64K (ROM)	HD6433040VTF	8	TQFP-100	3V	32K (ROM)

H8 / 3048



(*1)

Device	Memory
H8 / 3048	128K OTP 4K RAM
H8 / 3047	96K OTP / ROM 4K RAM
H8 / 3044	32K OTP / ROM 2K RAM

The H8/3048 has been developed as an application specific standard product for use in hand held battery driven applications which need a powerful microcontroller. This family also contains an industry leading 128KBytes of program memory and up to 4K of RAM.

Part Number	Speed (MHz)	Package	Voltage	ROM / RAM	OTP	Part Number	Speed (MHz)	Package	Voltage	ROM / RAM	OTP
HD6473048F	18, 16, 12, 10	QFP-100	5V	128K / 4K	✓	HD6433048F	18, 16, 12, 10	QFP-100	5V	128K / 4K	✗
HD6473048TF	18, 16, 12, 10	TQFP-100	5V	128K / 4K	✓	HD6433048TF	18, 16, 12, 10	TQFP-100	5V	128K / 4K	✗
HD6473048VF	13, 8	QFP-100	3V	128K / 4K	✓	HD6433048VF	13, 8	QFP-100	3V	128K / 4K	✗
HD6473048VTF	13, 8	TQFP-100	3V	128K / 4K	✓	HD6433048VTF	13, 8	TQFP-100	3V	128K / 4K	✗
HD6473047F	18, 16, 12, 10	QFP-100	5V	96K / 4K	✓	HD6433047F	18, 16, 12, 10	QFP-100	5V	96K / 4K	✗
HD6473047TF	18, 16, 12, 10	TQFP-100	5V	96K / 4K	✓	HD6433047TF	18, 16, 12, 10	TQFP-100	5V	96K / 4K	✗
HD6473047VF	13, 8	QFP-100	3V	96K / 4K	✓	HD6433047VF	13, 8	QFP-100	3V	96K / 4K	✗
HD6473047VTF	13, 8	TQFP-100	3V	96K / 4K	✓	HD6433047VTF	13, 8	TQFP-100	3V	96K / 4K	✗
HD6473044F	18, 16, 12, 10	QFP-100	5V	32K / 2K	✓	HD6433044F	18, 16, 12, 10	QFP-100	5V	32K / 2K	✗
HD6473044TF	18, 16, 12, 10	TQFP-100	5V	32K / 2K	✓	HD6433044TF	18, 16, 12, 10	TQFP-100	5V	32K / 2K	✗
HD6473044VF	13, 8	QFP-100	3V	32K / 2K	✓	HD6433044VF	13, 8	QFP-100	3V	32K / 2K	✗
HD6473044VTF	13, 8	TQFP-100	3V	32K / 2K	✓	HD6433044VTF	13, 8	TQFP-100	3V	32K / 2K	✗

H8 / 300H Support Tools

H8 / 300H C Compiler

Hitachi can support the H8/300H with a European developed ANSI C Compiler. This compiler produces highly optimised object code for all the different addressing options provided by the H8/300H CPU.

Data Types

Depending on the compiler switches used, the C Compiler (ICCH8300) will use either 16 or 32-bits integer types. The other data types and their corresponding ranges are shown in *Table 10.*

Table 10 -

Type	Size	Range
Char	1Byte	0 to 255 -128 to +127
Int (16-bit)	2Bytes	0 to 65,535 -32,768 to +32,767
Int (32-bit)	4Bytes	-2,147,483,648 to +2,147,483,647 0 to 4,294,967,295
Short	2Bytes	0 to 65,535 -32,768 to +32,767
Long	4Bytes	-2,147,483,648 to +2,147,483,647 0 to 4,294,967,295
Float	4Bytes	1.18E-38 to 3.39E+38
Double	8Bytes	2.23E-308 to 1.79E+308
Pointer	1-4Bytes	N/A

Pointers

To ensure efficient code is generated to access memory, ICCH8300 supports four types of pointers: tiny, near, far and huge. The addressing capability provided by each pointer type is shown in *Table 11.* Pointers can be classified in the source code to be of a certain size, or one of the memory models described in *Table 12* can be used to define a global pointer size.

Table 11 - H8 / 300H Pointer Types

Keyword	Storage in Bytes	Restrictions
tiny	1	May only point into short addressable area 0xFFFF00 to 0xFFFFFF
near	2	For H8/300 and H8/300H in minimum mode (-v0 and -v1): none For H8/300H in 1M and 16M mode (-v2 and -v3): may only access the area 0xFF8000 to 0x007FFF
far	4	The referenced object must reside entirely in one 64KBytes sgement
huge	4	No restrictions
tiny_func	1	May only point via the exception vector table
near_func	2	For H8/300 and H8/300H in minimum mode (-v0 and -v1): none For H8/300H in 1M and 16M mode (-v2 and -v3): may only access the area 0x0 to 0xFFFF

Interrupt Support

Using the interrupt pragma directive, functions to handle interrupts can be written in C. The compiler is shipped with header files which symbolically describe each device's interrupt vectors. In line functions are also provided so that interrupts can be efficiently and directly enabled or disabled from the C source.

In Line Functions

To cope with machine specific instructions, such as EEPROM data transfers and sleep operations, ICCH8300 provides a full set of in line functions (macros which substitute the function call with in line code). These are shown in *Table 13*.

I/O Access and Bit Operations

Microcontrollers typically need to spend a great deal of time manipulating I/O ports and

Table 12 - H8 / 300H Memory Models

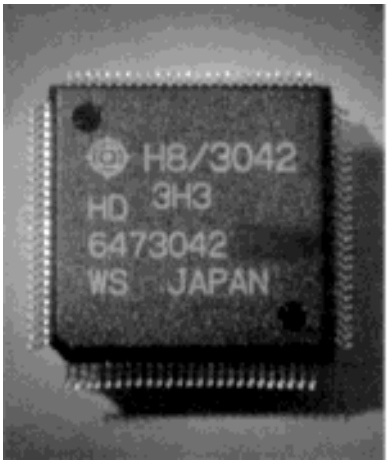
Model	Default Function Call	Default Data Type	Stack Size
Extra Small	far_func	far	256Bytes
Tiny	far_func	far	64KBytes
Mini	banked_func	far	64KBytes
Small	far_func	huge	64KBytes
Banked	banked_func	huge	16MBytes
Large	far_func	huge	16MBytes

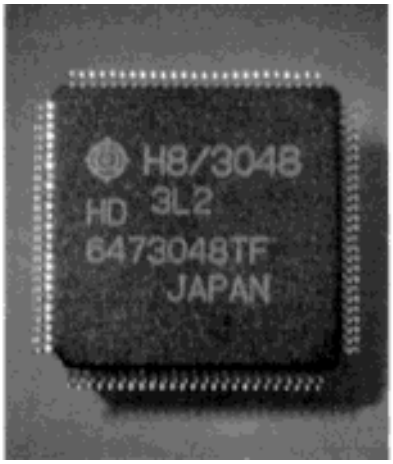
Normal Mode

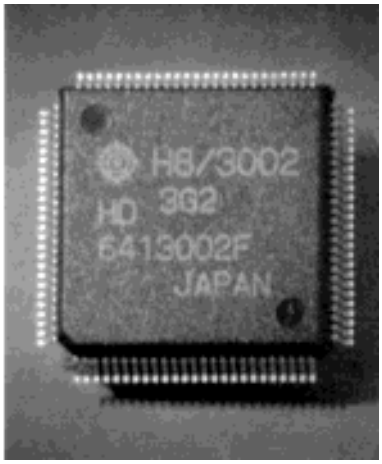
Model	Default Function Call	Default Data Type	Stack Size
Extra Small	tiny_func	near	256Bytes
Tiny	tiny_func	near	64KBytes
Mini	banked_func	near	256Bytes
Small	near_func	near	256Bytes
Banked	banked_func	near	64KBytes
Large	near_func	near	64KBytes

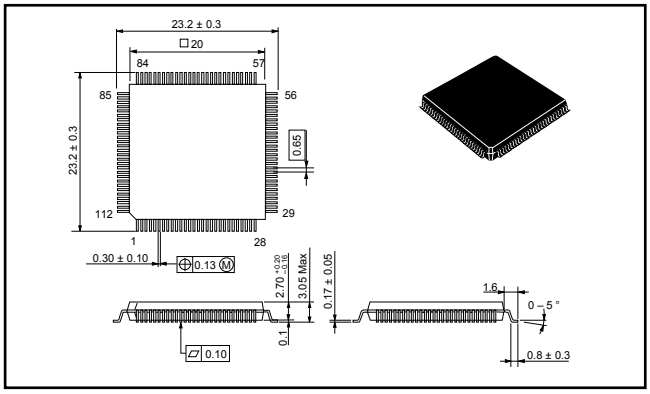
Advanced Mode

H8 / 300H Packages

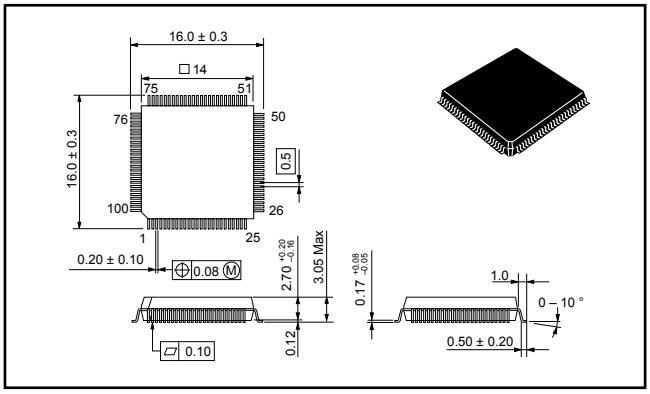




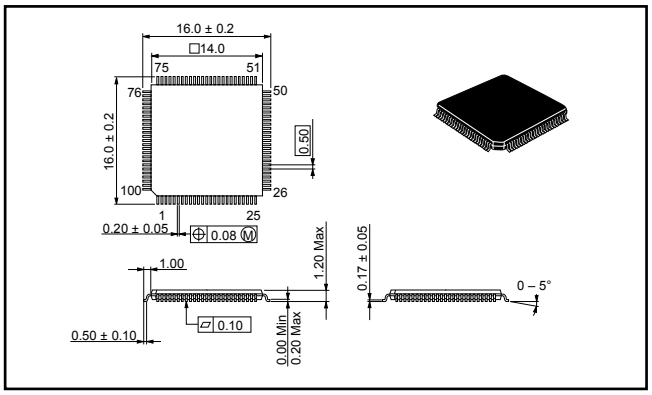




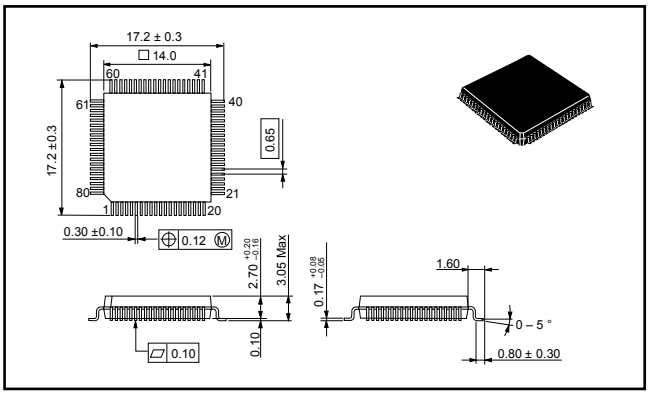
QFP-112



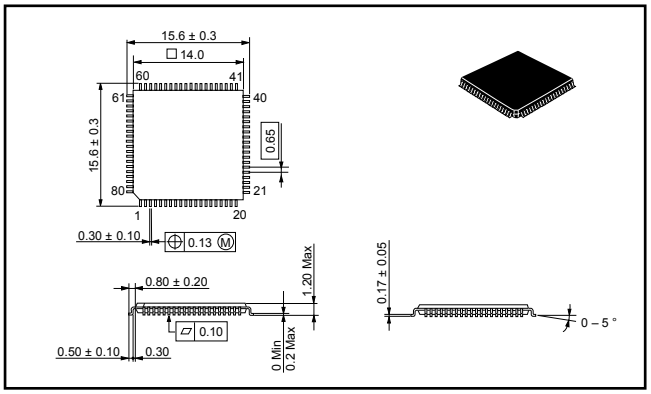
QFP-100B



TQFP-100B



QFP-80A



TQFP-80

Table 13 - H8 / 300H In Line Functions

Sleep	Execute Sleep Instruction
no_operation	No operation instruction
set_interrupt_mask	Load value (1 or 0) to interrupt mask
read_E_port	Move data in synchronisation with E clock (H8/300 only)
write_E_port	
do_byte_eepmov	Execute EEPROM move instruction
do_word_eepmov	
read_CCR	Read / write CCR
write_CCR	
and_CCR	Perform logical operations on CCR
or_CCR	
xor_CCR	

peripheral control registers, often in a bit wise manner, the H8/300H compiler has been optimised to perform these operations efficiently.

Peripheral registers (and thus I/O ports) can be accessed using simple cast statements. For bit addressable I/O locations the Special Function Register (SFR) keyword can be used as shown in *Figure 28*.

Utility Programs

The H8/300H compiler and assembler come complete with a powerful link utility (XLINK)and a librarian (XLIB). The linker allows the user to link multiple relocatable object files whilst performing stringent ANSI type checking. In addition it can be used to provide information on the actual memory allocation performed. The librarian allows the user to maintain and update the supplied system libraries in addition to creating new ones.

Figure 28 - Using SFR Bit Operations

```
#pragma language = extended

sfr SCI0_SSR = 0x ffffb4;  /* define bit variable at SSR bit 6  */

.
.
.

if (SCI_SSR.6)    /*  check Rx ready bit  */
  {
```

CIDE

The C level Integrated Debugging Environment tool (CIDE) provides a user-friendly debugging environment for the H8/300H family. CIDE is available for both the PCE and low cost evaluation boards, with a version also available to support the Hitachi real time Operating System (HIOS).

Source Level Debugging

CIDE allows the user to view the code currently under development in three formats. The full C source may be viewed, or a mixture between the C source and the associated assembler produced by the compiler, or symbolic assembler only (for each form of display: breakpoint and stepping utilities are level sensitive). Watchpoints may be set on variables at full C level, at symbolic level or at address level. Local variables may be watched when in scope.

Ordering Information

	PC	Sun 3	Sun 4	HP	VAX / VMS
Compiler, Assembler, Linker & Librarian	SE083HCPC	SE083HCSUN3U	SE083HCSUN4U	SE083HCHPU	SE083HCVAX
Assembler	SE083HPC	SE083HSUN3U	SE083HSUN4U	SE083HHPU	SE083HVAX

Emulator Features

When CIDE is running in conjunction with a PCE or E7000PC emulator, all the debugging features provided by the emulator can also be accessed by the user including the real time trace and full breakpoint facilities.

PCE8300H - PC Embedded In-Circuit Emulators

Features

- Real time in-circuit emulation of H8/300H series microcontrollers at 16MHz @ 5V and 8MHz @ 3.0V
- Parallel connection to IBM AT PC (compatible) for fast data transfer using swing buffering
- 2MBytes (zero wait state) user memory resident on-board, 1MBytes fixed and 1MBytes relocatable in four 256KBytes blocks
- Real time trace operational as a "rolling window" or as "address switchable"
- Complex breakpoint operation
- Windows environment frontend software with
  - Comprehensive command set
  - Fully symbolic debugging capability
  - Built-in assembler / disassembler
  - On-line context sensitive help
  - Command line operation for efficient use with command buffer
- Optional HLL Debug using CIDE

Introduction

The PCE8300H range of real time, in-circuit emulators can be used for the system development and debugging of the H8 / 300H microcontroller family applications. It is the latest product in the range of PC-based support tools for the H-Series microcontrollers and is designed with the aim of providing a high level of performance for minimum cost.

The PCE control software provides a powerful range of commands for controlling and interrogating the emulator hardware, through either "windows" or a command line on the host PC.

PCE8300H - Specification

Clock

On-board oscillators (user selectable) producing 16, 12, 10 or 8MHz internal system clock or target clock up to 16MHz.

Memory Size

- 2MBytes of emulation RAM with no wait states
- 1MBytes fixed from address H'000000 and 4 relocatable 256KBytes areas
- Base address of the relocatable areas can be set anywhere within the possible 16MBytes address range depending on the CPU mode

Memory Mapping

Memory is mapped as emulation RAM (internal), target system (user), write protected (ROM)or defined as not existing (guarded). Memory mapping is resolved to 32 word blocks.

Breakpoints

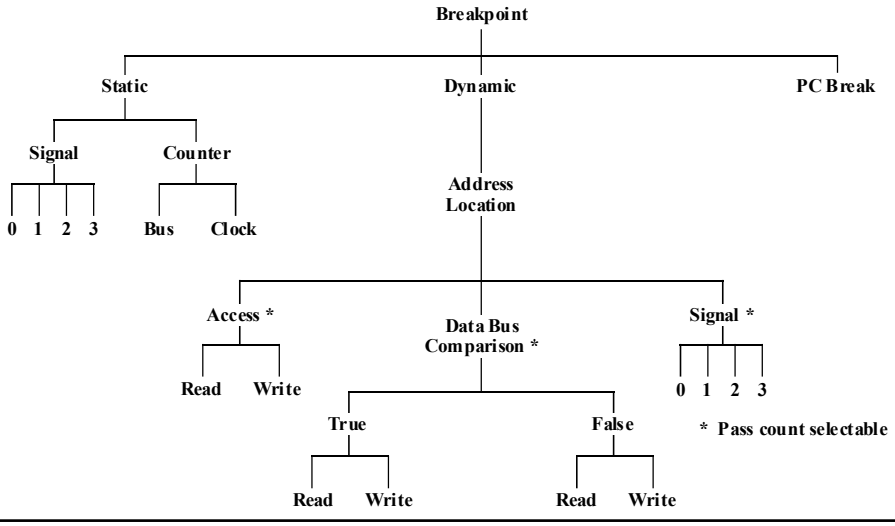
A complex breakpoint system allows a variety of conditions to be set up which, if encountered, will cause emulation to halt.

Three types of breakpoint are supported (see *Figure 29*).

Dynamic Breakpoints

Dynamic breakpoints are activated

Figure 29 - PCE8300H Breakpoints



by a condition on an access (read or write) to a certain address and can be qualified by either:

- Nothing (ie. all accesses)
- 8-bit compare (true / false)
- 16-bit compare (true / false)
- One of 4 external probe signals (low level)

A pass count can be specified to a dynamic breakpoint. There are two 12-bit counters and two 16-bit comparators available to all the breakpoint logic.

Dynamic breakpoint coverage is limited to one 1MBytes area fixed from H'000000 in word boundaries and four relocatable 256KBytes areas configurable within the possible 16MBytes address range.

Program Counter Breakpoints

These are used on op-code reads and halt the program if the op-code would have been executed. Program Counter breakpoints cover the whole 16MBytes address range for word accesses. These can also be set within ROM areas even on a target system.

Static Breakpoints

Only one static breakpoint can be set. Program execution is halted after a specified number of bus or clock cycles, or on the unconditional occurrence of an external probe signal.

Dynamic and static breakpoints can be given a priority level.

- Always - causes a halt as soon as condition is met (default)
- Level 1 - will not cause a program halt but is used to qualify a level 2 breakpoint
- Level 2 - causes a program halt if a level 1 has already been detected

Any level 1 breakpoint can qualify a level 2 breakpoint if more than one level 1 breakpoint is set.

Program execution can also be halted if an access to a guarded area is detected or a write cycle to a read only location occurs. These breakpoints can be ignored if required.

There are two 12-bit counters and two 16-bit comparators available to the breakpoint logic. These can enable complex pass-counted breakpoint operations, for example, to be set within nested loops.

Real Time Trace

The PCE8300H trace facility allows examination of the processor activity prior to program execution being terminated. Up to 32K bus cycles can be stored, either a a "rolling buffer" where the last 32K cycles are stored or "address selectable" which can be used for tracing subroutines and procedures. The traced signals include the address and the data buses along with processor status / control signals and the user probe signals.

The trace buffer can be displayed in two forms. As raw unprocessed data or as disassembled instructions incorporating any defined symbols.

Symbolic

Full symbolic debugging is possible with all relevant commands accepting symbol entries including the line assembler and disassembler.

Host Interface

A small PC interface card plugs into a standard PC-compatible XT / AT card slot and provides an 8-bit parallel interface to the IBM bus. This is connected to the main PCE board via a ribbon cable. The PC / PCE interface consists of a DPRAM that sits in the PC memory map and will operate as a double buffer facilitating fast program transfer. The PCE interface card address range is address selectable so as not to conflict with other cards the user may already be using. Power is supplied from the PC so no auxilliary power supply is necessary.

Other Features

- On-line context sensitive help eases use.
- Windows-based front end software
- Full circuit protection when connected to user hardware
- DC-DC converter circuit for 3.3V target operation
- Durable target cable
- Run time clock giving emulation time (accuracy 0.25µs to 814 days)
- Oscilloscope trigger facility
- Batch file execution
- Session logging
- Limited performance analysis

A block diagram of the PCE8300H is shown in *Figure 30*.

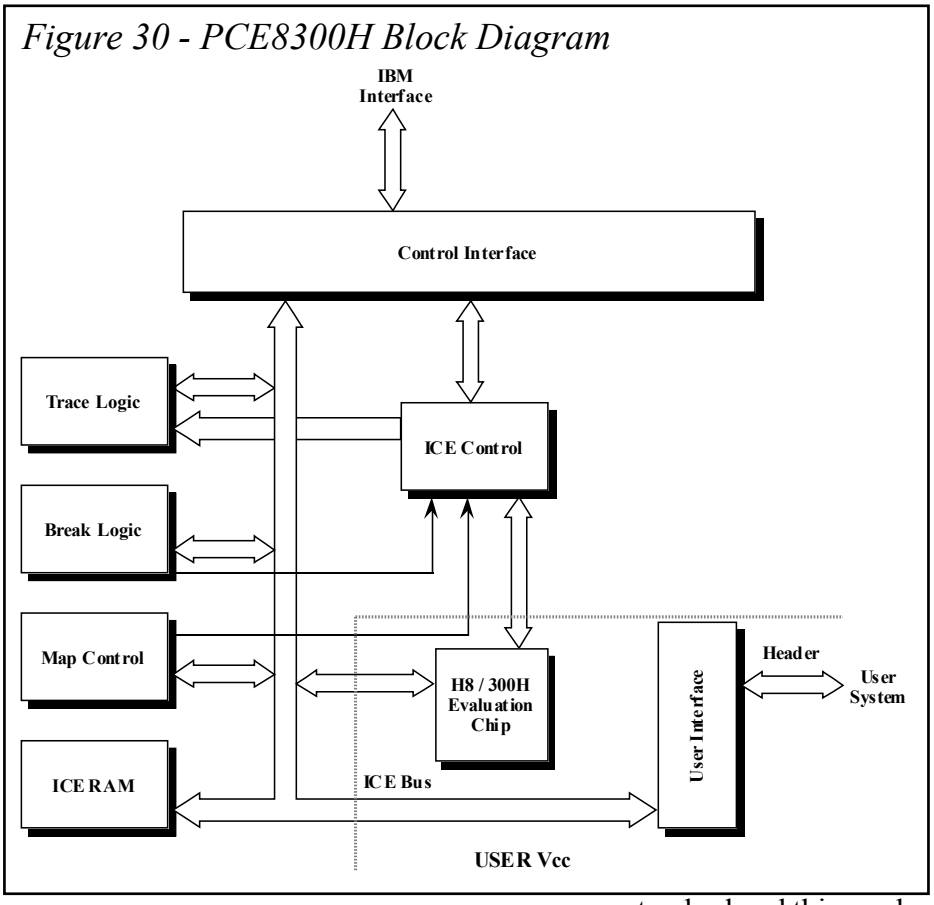
Power

5V or 3.3V derived from PC power supply (with no target hardware)

Size

- Interface card 150mm x 100mm approximately (reduced PC XT format)
- PCE8300H: approximately extended double EuroCard size

Figure 30 - PCE8300H Block Diagram



Processors Supported
H8/3001, H8/3002, H8/3003, H8/3042, H8/3048

E7000 Emulator

The E7000 series of in-circuit emulators provides a very high level of debugging capability to users of personal computers and engineering workstations. They have been designed to work in conjunction with PCs via a high speed parallel interface or a workstation via a ethernet connection. Optional graphical user interfaces are available for Windows on the PC and for UNIX on SUN and HP workstations.

Emulation Features

The E7000 systems come with 256KBytes of real time emulation memory as standard and this can be upgraded to 4MBytes by adding optional memory cards. The emulation memory can be mapped to meet target requirements in blocks of 8KBytes. Memory can be mapped onto the user's target, into emulator memory, as write protected and as guarded.

Two types of breakpoint are supported by the E7000, Program Counter (PC) and hardware condition.

The PC breakpoint stops program execution when code from a specified address or range of addresses is executed. A pass count can also be specified. This function is performed by hardware so PC breaks can be set in an area of ROM on the user's target system. A total of 255 PC breaks can be set at any one time.

The hardware condition break allows a specified combination of data bus, address bus, control signals, interrupt inputs and external probe conditions to halt program execution. For example, if a break is required after the serial port is written to 20 times with the value 22, then these breakpoints will perform this function. Breakpoint sequencing can also be done by the hardware condition breaks. Four of this type of breakpoint can be set at any time.

A 32K cycle trace buffer is provided by the E7000. The trace stores a comprehensive set of data from each bus cycle, including the bus values, control signals, interrupts, external probes and Vcc.

The information can be displayed in a disassembled format or as straight bus cycles.

The mode of trace acquisition can be specified to collect all data or to collect specified data, such as all data within the execution of a subroutine. It is also possible to trigger the trace using external signals.

Performance Analysis and Code Coverage

In high throughput microcontroller systems, it is often desirable to capture information which allows the user to judge the performance of sections of the code so decisions can be taken as to where to spend effort in code optimisation. This requirement is met using the performance analysis function of the E7000.

Using this function the amount of time taken in the execution of up to

four subroutine areas can be displayed. The measured information shows the number of times the subroutine is executed plus the total time taken by each subroutine.

The E7000 also allows the user to perform coverage checking on a systems code. Using this facility it is possible to show which areas of memory have been executed from within a 2MByte memory area. This enables the identification of any untested code within the system.

User Interface

The user interface for the E7000 emulator is CIDE under Windows on the PC and a high functionality GUI is supplied for use on SUN or HP workstations. Both of these interfaces provide full high level language debugging, including the display of register based local variables and array or structure elements.

System Kits

Hitachi can supply several system kits, which include all the items needed to start development with the H8/300H.

Ordering Information

PCE's (Emulator + Assembler Debugger)

Device/ Package		H8/3003	H8/3002	H8/3001	H8/3042,1,0	H8/3032,1,0	H8/3048,7,4
QFP-80A1	s	●	●	PCE3001Q80A1 (PCE3003 + PHB3001Q80A1)	●	●	●
	d						
TQFP-80C1	s	●	●	PCE3001T80C1 (PCE3003 + PHB3001T80C1)	●	●	●
	d						
QFP-80A2	s	●	●	●	●	PCE3032Q80A2 (PCE3003 + PHB3032Q80A2)	●
	d						
TQFP-80C2	s	●	●	●	●	PCE3032T80C2 (PCE3003 + PHB3032T80C2)	●
	d						
QFP-100A	s	●	●	●	PCE3042Q100A (PCE3003 + PHB3042Q100A) (PCE3048 + PHB3042Q100A)	●	●
	d						
	d						
QFP-100B	s	●	PCE3002Q100B (PCE3003 + PHB3002Q100B)	●	PCE3002Q100B (PCE3003 + PHB3002Q100B) (PCE3048 + PHB3002Q100B)	●	PCE3048Q100B (PCE3048 + PHB3048Q100B)
	d						
	d						
QFP-112	s	PCE3003Q112 (PCE3003 + PHB3003Q112)	●	●	●	●	●
	d						

S5's (PCE + C Compiler + CIDE)

Device/ Package		H8/3003	H8/3002	H8/3001	H8/3042,1,0	H8/3032,1,0	H8/3048,7,4
QFP-80A1	s	●	●	S5-3001Q80A1 (S5-3003 + PHB3001Q80A1)	●	●	●
	d						
TQFP-80C1	s	●	●	S5-3001T80C1 (S5-3003 + PHB3001T80C1)	●	●	●
	d						
QFP-80A2	s	●	●	●	●	S5-3032Q80A2 (S5-3003 + PHB3032Q80A2)	●
	d						
TQFP-80C2	s	●	●	●	●	S5-3032T80C2 (S5-3003 + PHB3032T80C2)	●
	d						
QFP-100A	s	●	●	●	S5-3042Q100A (S5-3003 + PHB3042Q100A) (S5-3048 + PHB3042Q100A)	●	●
	d						
	d						
QFP-100B	s	●	S5-3002Q100B (S5-3003 + PHB3002Q100B)	●	S5-3002Q100B (S5-3003 + PHB3002Q100B) (S5-3048 + PHB3002Q100B)	●	S5-3048Q100B (S5-3048 + PHB3048Q100B)
	d						
	d						
QFP-112	s	S5-3003Q112 (S5-3003 + PHB3003Q112)	●	●	●	●	●
	d						

Notes:
s Single order placement for this device/package
d Double order placement for this device/package using generic emulators & header assemblies
● Invalid device/package option

Ordering Information

E7000PC's (Emulator + CIDE Debugger)

Device/ Package		H8/3003	H8/3002	H8/3001	H8/3042,1,0	H8/3032,1,0	H8/3048,7,4
QFP-80A1	s	●	●	E73001Q80A1 (HS3001ECH71H)	●	●	●
	h						
TQFP-80C1	s	●	●	E73001T80C1 (HS3001ECN71H)	●	●	●
	h						
QFP-80A2	s	●	●	●	●	E73032Q80A2 (HS3032ECH71H)	●
	h						
TQFP-80C2	s	●	●	●	●	E73032T80C2 (HS3032ECN71H)	●
	h						
QFP-100A	s	●	●	●	E73042Q100A (HS3042ECF71H)	●	●
	h						
QFP-100B	s	●	E73002Q100B (HS3042ECH71H)	●	E73002Q100B (HS3042ECH71H)	●	E73048Q100B (HS3042ECH71H)
	h						
QFP-112	s	E73003Q112 (HS3003ECH71H)	●	●	●	●	●
	h						

S6's (E7000PC + C Compiler + CIDE)

Device/ Package		H8/3003	H8/3002	H8/3001	H8/3042,1,0	H8/3032,1,0	H8/3048,7,4
QFP-80A1	s	●	●	S6-3001Q80A1 (HS3001ECH71H)	●	●	●
	h						
TQFP-80C1	s	●	●	S6-3001T80C1 (HS3001ECN71H)	●	●	●
	h						
QFP-80A2	s	●	●	●	●	S6-3032Q80A2 (HS3032ECH71H)	●
	h						
TQFP-80C2	s	●	●	●	●	S6-3032T80C2 (HS3032ECN71H)	●
	h						
QFP-100A	s	●	●	●	S6-3042Q100A (HS3042ECF71H)	●	●
	h						
QFP-100B	s	●	S6-3002Q100B (HS3042ECH71H)	●	S6-3002Q100B (HS3042ECH71H)	●	S6-3048Q100B (HS3042ECH71H)
	h						
QFP-112	s	S6-3003Q112 (HS3003ECH71H)	●	●	●	●	●
	h						

Notes:
s Single order placement for this device/package
h Header assembly for this package/processor type
● Invalid device/package option